The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! Sept. 1987: Volume 8, Issue 02. Canada $4.25 USA $3.50

Operating Systems

The 32-bit Amiga: Upgrade to a 68010
Amiga Messages, Ports and Signals
Kernal LISTEN and its Relatives
Garbage Collector Revealed
C64 Mouse Driven Menus
Mr Ed: A C64 Text Editor
CBM RAM Cartridges
SYS 65478 Revisited
Disk Error Recovery
C128 Function Keys
Amiga Dispatches
In the CP/M Mode
CP/M User Areas
Mandelbrot Halo
Plus New Hit Single:
Makin' The Transactor

FREE! T-Shirt Offer
see page 75

THE TIME SAVER

J. MOSTACCI

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $8.95 US, $9.95 Cdn. Per Issue
6 Disk Subscription (one year)
Just $45.00 US, $55.00 Cdn.
(see order form at center fold)

Now Amiga Owners Can Save Time Too!

Transactor Amiga Disk #1, $12.95 US, $14.95 Cdn.

All the Amiga programs from the magazine, with complete documentation on disk, plus our pick of the public domain!

Volume 8
Issue 02
Paid Circulation
15,000

Operating Systems

Start Address Editorial 3

Bits and Pieces . . . 6
The Jiggler
The Striped Crawler That Drips Blood and Kills People
Resets Revisited
1541 + 1702 = #?$•!!
Do You Know Where Your Head Is?
More 1541 Tips
Break C64
128 Notepad
Save C-128 Variables in RAM Expansion
128 BASIC Linefinder
No-line LIST For the 128
Table Look-Up Without Arrays
Static Detector
Marble Madness Teamwork Tip
Simple C64 Hi-Res Printer Dump
Ribbon Alternative
Formatting An Un-Notched Disk
Protect Those Vectors!
Sorting On The Fly

Letters 12
Amiga coverage unjust
Ingratitude, publicly expressed
Bang-bang floppy-copy
Explaining the Drivelight Zone
Boat leaves Commodore stranded
Bird, plane, or Commodore 64?
Plus/4 Tech Info Source
Communicating Braces
Getting Poor Quick
Another IEEE Interface for the C-128
More Quicksilver IEEE Info Wanted
Super-C 3.0 Fix Available

News BRK 75
Our New Home
Advertisers Wanted
New Canadian Prices
Cover Price Increase
Shipping Fee on Mail Orders
Don't Forget the Sales Tax!
Sign Of The Times
Dealer Inquiries Welcome
Group Subscription Rates: The 20/20 Deal
T-Shirt Offer Continues
Mail-Order Products No Longer Offered
New Mail-Order Products
The Bits and Pieces Disk
Bits Book AND Disk
The Amiga Disk is here!
The Potpourri Disk
TransBASIC II
The Glink is Back!
New Set of Microfiche
Transactor Disks, Back Issues, and Microfiche
Portland Company Vanishes
4040 Drive Internals
CAD for the Amiga
B.E.S.T.Business Management
Public Domain Programs
The New PAL JR.
NLQ for the Gemini 10X
Supradrive Amiga Hard disk
Auto Disk Menu/Program Loader
A-Talk Communication Tools for the Amiga

TransBloopurz . . . 11
Vol8 Iss.1: Strange Cases of Backwards Braces
Vol7 Iss.6: EPROM Programmer Update
Vol7 Iss.6: Textscan
Vol7 Iss.3: Keyboard Expander
Vol6 Iss.6: VARPTR

TeleColumn 16
Mouse Driven Menus The power of an Atari ST on your C64! 17
Garbage Collector Revealed Crashed or collecting? Wonder no more! . . . 30
SYS 65478 A new look at an old dog 33
Kernal LISTEN and its Relatives More control, less code 36
CBM RAM Cartridges Use them from Basic on the C64 38
In the CP/M Mode How to get more out of C128 CP/M 42
CP/M User Areas Making the most of practically nothing 46
Disk Error Recovery Two nifty ML subroutines 49
The 32-bit Amiga Enhance your Amiga's power with the MC68010 50
Messages, Ports and Signals An Amiga conversation piece 53
Amiga Dispatches Our plugged-in columnist brings you the latest news 60
A C64 Text Editor Complete with source, of course, of course 62
Mandelbrot Halo Exploring the Mandelbrot Set on the C128 68
C128 Function Keys All you'll ever need to know 74

Note: Before entering programs, see "Verifizer" on page 4

The Transactor
The Tech/News Journal For Commodore Computers

Editor-in-Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Online Editor
Nick Sullivan

Cover Art
Jason Goldberg

Administration & Subscriptions
Jennifer Reddy

Contributing Writers

Ian Adam
David Archibald
Jim Barbarello
Anthony Bertram
Tim Bolbach
Ranjan Bose
Donald Branson
Anthony Bryant
Jim Butterfield
Dale A. Castello
Betty Clay
Joseph Caffrey
Tom K. Collopy
Robert V. Davis
Elizabeth Deal
Frank E. DiGioia
Chris Dunn
Michael J. Erskine
Jack Farrah
William Fossett
Jim Frost
Miklos Garamszeghy
Eric Germain
Michael T. Graham
Eric Guiguere
Thomas Gurley
R. James de Graff
Tim Grantham
Adam Herst
Thomas Henry
John Holttum
John Houghton
Robert Huehn
David Jankowski
Clifton Karnes
Lorne Klassen
Jesse Knight
Gregory Knox
David Lathrop
James A. Lisowski
Richard Lucas
Scott Maclean
David Martin
Steve McCrystal
Stacy McInnis
Chris Miller
Terry Montgomery
Ralph Morrill
Rick Morris
Michael Mossman
Bryce Nesbitt
Gerald Neufeld
Noel Nyman
Kevin O'Connor
Richard Perrit
Terry Pridham
Raymond Quirling
Richard Richmond
Gary Royal
John W. Ross
Dan Schein
E.J. Schmahl
David Shiloh
John Spencer
Darren J. Spruyt
Aubrey Stanley
David Stidolph
Richard Stringer
Anton Treuenfels
Audrys Vilkas
Jack Weaver
Evan Williams

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') is a straight line as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print ' ' flush right ' ' – would be shown as – print ' '[10 spaces]flush right ' '

Cursor Characters For PET / CBM / VIC / 64

Down – [q]
Up – [Q]
Right – [ ]
Left – [Lft]
RVS – [r]
RVS Off – [R]
Insert – [ ]
Delete – [ ]
Clear Scrn – [ ]
Home – [s]
STOP – [ ]

Colour Characters For VIC / 64

Black – [P]
White – [ ]
Red – [ ]
Cyan – [Cyn]
Purple – [Pur]
Green – [ ]
Blue – [ ]
Yellow – [Yel]
Orange – [A]
Brown – [U]
Lt. Red – [V]
Grey 1 – [W]
Grey 2 – [X]
Lt. Green – [Y]
Lt. Blue – [Z]
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – [E]
F2 – [ ]
F3 – [F]
F4 – [ ]
F5 – [G]
F6 – [K]
F7 – [H]
F8 – [ ]

Please Note: The Transactor's
***NEW* NEW**
phone number is: (416) 764-5273

CompuServe Accounts

Contact us anytime on GO CBMPRG, GO CBMCOM, or EasyPlex at:

Karl J.H. Hildon	76703,4242
Richard Evers	76703,4243
Chris Zamara	76703,4245
Nick Sullivan	76703,4353

The Transactor is published bi-monthly by Transactor Publishing Inc., 85 West Wilmot Street, Unit 10, Richmond Hill, Ontario, L4B 1K7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, P.O. Box 338, Station C, Buffalo, NY, 14209 **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64, 128, Amiga) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $19 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 85 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7, 416 764 5273. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Quantity Orders:

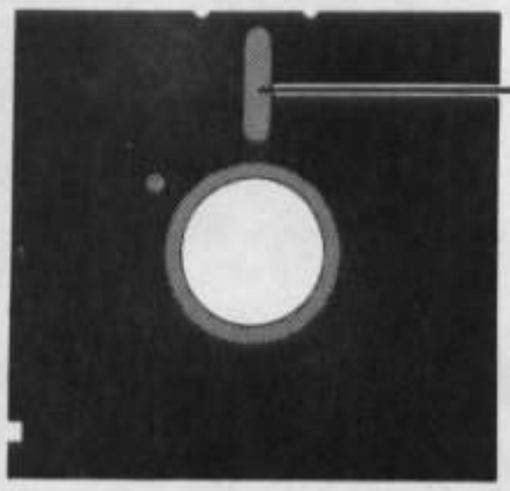
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 03, 04, 05, 06, and Vol 5 Issues 02, 03, 04 are available on microfiche only
Still Available: Vol. 4: 01, 02. Vol. 5: 01, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06. Vol. 7: 01, 02, 03, 04, 05, 06. Vol. 8: 01, 02

Back Issues: $4.50 each. Order all back issues from Richmond Hill HQ.

Editorial contributions are always welcome. Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Solicited material is accepted on an all rights basis only. Write to the Richmon Hill address for a writers package. The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

Makin' The Transactor

Whew! I can't believe it's time for another editorial page. Since writing my last one the hands on the clock have been spinning like a crankshaft. I know, I know, you're thinking, "it seems like every second editorial he writes more tales of countless hours of sweat and toil, and it sounds like he's about to plunge into another one.". Well, I am, but it's not a sad song, quite the contrary. And it's not just me. . . we've all been chasing our tails over the last 10 weeks.

Let me tell ya 'bout it!

Sittin' in the classroom, thinkin' it's a drag,
Listenin' to the teacher, well that just ain't my bag,

. . . oops, wrong lyrics. Oh, here they are, but the melody is the same (Smokin' in the Boys Room by Brownsville Station):

Negotiated terms, agreed to take the rap,
Transactor would be sold to us for three dollars cash,
But computers and assets, and outstanding debts,
Would add up to more than we would ever expect.

Signed all the papers, Shook all their hands,
Said see ya later, And packed up the van,
Headed off to Richards, With one load then two,
2 o'clock A.M., and you know what we would do?. . . We'd go. . .

Makin' The Transactor
Makin' The Transactor
Well readers don't ya laugh about writin' this tune,
'Cause everybody knows that Transactor must be out by June.

Next day would come, time to check out the shop,
Construction hadn't started, but there's no time to stop,
Called up the landlord, "hey what's happenin' man?",
"If we don't move in soon, it's gonna mess up our plan, for

Program after program, loadin' up to CompuServe,
My faulty VDT is gonna kill my optic nerve,
Prepared 2 meg of text, just 3 more meg to go,
By the time the DA's done we'll probably be havin' snow. And we'll be. . .

(Chorus)

Should we do this show, called Computer Expo?,
Or should we get the mail sittin' in Buffalo?,
End up doin' both, made G-Links 'till three,
And all to find some kids went on a T-Shirt stealing spree.

Our place is almost ready, it's almost time to move
Our old lease is up, so we gotta do it soon,
Better paint the warehouse first, to keep the dust down,
Y'know if we don't, we'll be cleanin' all around. . . Before we go. .

Rented out a truck to move 10 tons of magazines,
There was desks and chairs and other stuff packed in just like sardines,
the packages were heavy; they made our muscles strain,
Heavin' the 50 pounders way up top sure was a pain. Then we went. . .

(Chorus) *("sackbut, take it Benny. . .")*

Richard was busy, with his own set of probs,
He had to get things going on a number of jobs,
There was Lawyers, and Bankers, and Fishermen too,
Keepin 'em all happy was like workin' in a zoo.

Mastercard, Visa, and old mother Bell,
Were takin' their time getting ready as well,
Customers are calling, their orders aren't filled
"I'd love to help you sir, but Visa must be billed."

The top floor was unfinished, it wasn't in the deal
It saved a bit of money and we thought, "no big ordeal",
First there was sanding, then painting with blue
The overspray gave me an unnatural hue.

With Jim as the foreman, the carpet got laid,
Then Rick came around and the counter got made,
Just in time too, for our opening due,
The party is tommorrow but there's still more to do.

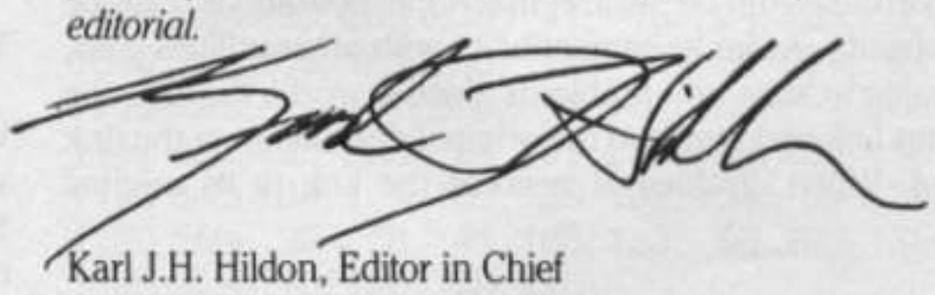
The counter was up, the equipment was not,
We loaded Rick's truck with an incredible lot,
Grabbed the computers, the stereo, and TV,
And were sure that on the way, we'd be charged with B 'n' E. Jailed and. .

(Chorus)

With the place all set up, it was time for the bash,
We brought in the liquor and took out the trash,
People came from all around, there's some hungover yet,
"Thanks for comin' folks, now we've got some type to set".

'Cause we're. . .
Late with The Transactor
Late with The Transactor
Now readers I may not be that good with a rhyme,
But now you all know why this Transactor won't be out on time.

(Spoken) Tune in next issue, same time, same place, for a more serious editorial.



Karl J.H. Hildon, Editor in Chief
with help from Nick and Chris. . . thanks guys!

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are five versions of VERIFIZER here; one for PET/CBMs, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum error, re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 4096 to enable the Plus 4 version (off: SYS 4099)
SYS 3072,1 to enable the C128 version (off: SYS 3072,0)
BANK 15: SYS 1024 for B128 (off: BANK 15: SYS 1027)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifizer could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI  10 rem* data loader for "verifizer 4.0" *
CF  15 rem pet version
LI  20 cs=0
HC  30 for i=634 to 754:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
OG  60 if cs<>15580 then print "***** data error *****": end
JO  70 rem sys 634
AF  80 end
IN  100 :
ON  1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB  1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK  1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB  1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE  1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI  1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB  1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA  1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE  1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL  1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA  1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI  1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB  1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM  1130 data 251, 133, 251,  96
```

VIC/C64 VERIFIZER

```
KE  10 rem* data loader for "verifizer" *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print "***** data error *****": end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

VIC/64 Double Verifizer **Steven Walley, Sunnymead, CA**

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifizer-displayed codes. DOUBLE VERIFIZER solves that problem by showing the two-letter verifizer code on both the first and second row of the TV screen. Just run the below program once the regular Verifizer is activated.

```
KM 100 for ad=679 to 720:read da:poke ad,da:next ad
BC 110 sys 679: print: print
DI 120 print "double verifizer activated":new
GD 130 data 120, 169, 180, 141,  20,   3
IN 140 data 169,   2, 141,  21,   3,  88
EN 150 data 96,  162,   0, 189,   0, 216
KG 160 data 157,  40, 216, 232, 224,   2
KO 170 data 208, 245, 162,   0, 189,   0
FM 180 data   4, 157,  40,   4, 232, 224
LP 190 data   2, 208, 245,  76,  49, 234
```

VERIFIZER For Tape Users **Tom Potts, Rowley, MA**

The following modifications to the Verifizer loader will allow VIC and 64 owners with Datasettes to use the Verifizer directly (without the loader). After running the new loader, you'll have a special copy of the Verifizer program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/64 VERIFIZER loader:

```
NB  30   for i=850 to 980: read a: poke i,a
AL  60   if cs<>14821 then print"*****data error*****": end
IB  70   rem sys850 on, sys853 off
--  80   delete line
--  100  delete line
OC 1000 data  76,  96,   3, 165, 251, 141,   2,   3, 165
MO 1030 data 251, 169, 121, 141,   2,   3, 169,   3, 141
EG 1070 data 133,  90,  32, 205,   3, 198,  90,  16, 249
BD 2000 a$="verifizer.sys850[space]"
KH 2010 for i=850 to 980
GL 2020 a$=a$+chr$(peek(i)): next
DC 2030 open 1,1,1,a$: close 1
IP 2040 end
```

Now RUN, pressing PLAY and RECORD when prompted to do so (use a rewound tape for easy future access). To use the special Verifizer that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifizer loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIZER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash. If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Plus 4 VERIFIZER

```
NI 1000 rem * data loader for "verifizer +4"
PM 1010 rem * commodore plus/4 version
EE 1020 graphic 1: scnclr: graphic 0: rem make room for code
NH 1030 cs=0
JI 1040 for j=4096 to 4216: read x: poke j,x: ch=ch+x: next
AP 1050 if ch<>13146 then print "checksum error": stop
NP 1060 print "sys 4096: rem to enable"
JC 1070 print "sys 4099: rem to disable"
ID 1080 end
PL 1090 data  76,  14,  16, 165, 211, 141,   2,   3
CA 1100 data 165, 212, 141,   3,   3,  96, 173,   3
OD 1110 data   3, 201,  16, 240,  17, 133, 212, 173
LP 1120 data   2,   3, 133, 211, 169,  39, 141,   2
EK 1130 data   3, 169,  16, 141,   3,   3,  96, 165
```

```
DI 1140 data  20, 133, 208, 162,   0, 160,   0, 189
LK 1150 data   0,   2, 201,  48, 144,   7, 201,  58
GJ 1160 data 176,   3, 232, 208, 242, 189,   0,   2
DN 1170 data 240,  22, 201,  32, 240,  15, 133, 210
GJ 1180 data 200, 152,  41,   3, 133, 209,  32, 113
CB 1190 data  16, 198, 209,  16, 249, 232, 208, 229
CB 1200 data 165, 208,  41,  15,  24, 105, 193, 141
PE 1210 data   0,  12, 165, 208,  74,  74,  74,  74
DO 1220 data  24, 105, 193, 141,   1,  12, 108, 211
BA 1230 data   0, 165, 210,  24, 101, 208, 133, 208
BG 1240 data  96
```

C128 VERIFIZER (40 column mode)

```
PK 1000 rem * data loader for "verifizer c128"
AK 1010 rem * commodore c128 version
JK 1020 rem * use in 40 column mode only!
NH 1030 cs=0
OG 1040 for j=3072 to 3214: read x: poke j,x: ch=ch+x: next
JP 1050 if ch<>17860 then print "checksum error": stop
MP 1060 print "sys 3072,1: rem to enable"
AG 1070 print "sys 3072,0: rem to disable"
ID 1080 end
GF 1090 data 208,  11, 165, 253, 141,   2,   3, 165
MG 1100 data 254, 141,   3,   3,  96, 173,   3,   3
HE 1110 data 201,  12, 240,  17, 133, 254, 173,   2
LM 1120 data   3, 133, 253, 169,  38, 141,   2,   3
JA 1130 data 169,  12, 141,   3,   3,  96, 165,  22
EI 1140 data 133, 250, 162,   0, 160,   0, 189,   0
KJ 1150 data   2, 201,  48, 144,   7, 201,  58, 176
DH 1160 data   3, 232, 208, 242, 189,   0,   2, 240
JM 1170 data  22, 201,  32, 240,  15, 133, 252, 200
KG 1180 data 152,  41,   3, 133, 251,  32, 135,  12
EF 1190 data 198, 251,  16, 249, 232, 208, 229,  56
CG 1200 data  32, 240, 255, 169,  19,  32, 210, 255
EC 1210 data 169,  18,  32, 210, 255, 165, 250,  41
AC 1220 data  15,  24, 105, 193,  32, 210, 255, 165
JA 1230 data 250,  74,  74,  74,  74,  24, 105, 193
CC 1240 data  32, 210, 255, 169, 146,  32, 210, 255
BO 1250 data  24,  32, 240, 255, 108, 253,   0, 165
PD 1260 data 252,  24, 101, 250, 133, 250,  96
```

B128 VERIFIZER **Elizabeth Deal, Malvern, PA**

```
1 rem save"@0:verifizerb128",8
10 rem* data loader for "verifizer b128" *
20 cs=0
30 bank 15:for i=1024 to 1163:read a:poke i,a
40 cs=cs+a:next i
50 if cs<>16828 then print"** data error **": end
60 rem bank 15: sys 1024
70 end
1000 data  76,  14,   4, 165, 251, 141, 130,   2, 165, 252
1010 data 141, 131,   2,  96, 173, 130,   2, 201,  39, 240
1020 data  17, 133, 251, 173, 131,   2, 133, 252, 169,  39
1030 data 141, 130,   2, 169,   4, 141, 131,   2,  96, 165
1040 data   1,  72, 162,   1, 134,   1, 202, 165,  27, 133
1050 data 233,  32, 118,   4, 234, 177, 136, 240,  22, 201
1060 data  32, 240,  15, 133, 235, 232, 138,  41,   3, 133
1070 data 234,  32, 110,   4, 198, 234,  16, 249, 200, 208
1080 data 230, 165, 233,  41,  15,  24, 105, 193, 141,   0
1090 data 208, 165, 233,  74,  74,  74,  74,  24, 105, 193
1100 data 141,   1, 208,  24, 104, 133,   1, 108, 251,   0
1110 data 165, 235,  24, 101, 233, 133, 233,  96, 165, 136
1120 data 164, 137, 133, 133, 132, 134,  32,  38, 186,  24
1130 data  32,  78, 141, 165, 133,  56, 229, 136, 168,  96
1140 data 170, 170, 170, 170
```

b i t s

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits column, we'll credit you in the column and send you a free one–year's subscription to The Transactor

Ok, we've got a great bunch of bits for you this time! We start with a couple of screen blitzes for the 64, the kind of program that made the bits section famous.

The Jiggler **Loren Teillon, Virginia Beach, VA**

Make your C64 "Twist and Shout" – just try this simple program and see what happens.

```
100 for t=0 to 15: poke 53270,t: next: goto 100
```

Just remove the GOTO to use this in your programs. Experiment with the for–next values for different effects. Great for games.

The Striped Crawler That Drips Blood and Kills People **Martin Spencer Brampton, Ontario**

Ooh, a real scary one this is, kids! Ok, so it doesn't drip blood. Well, it doesn't kill people either. Ok, maybe it isn't that scary, but look at those stripes in the border!

```
10 a=53280:b=0:c=11:d=12:e=15
20 pokea,b:gosub30:pokea,l:gosub30:pokea,d
   :gosub30:pokea,e:gosub30:goto20
30 ::return
```

(Make sure the two colons precede the RETURN on line 30.)

Ahem. Now on to some slightly more serious topics.

Resets Revisited **Keith Hendren Kelvington, Saskatchewan**

The Commodore 64C has an advantage over the older 64 in the way the reset lines are set up. In the newer model, a reset in the serial port will not reset the computer; similarly a reset in the user or expansion port will not reset the serial line. This gives us the opportunity to install two reset switches – one between pin 3 of the user port and ground, the other between pin 6 of the serial port and ground. (See the Commodore 64 Programmers Reference Guide or The Complete Commodore Inner Space Anthology for numbering of pins.) There is plenty of room above the cassette port in the back of the lower half of the computer case for the SPST momentary contact switches. Here they are fairly immune to accidental operation, and don't prevent easy removal of the top of the keyboard.

There are several advantages to having two reset switches. In many cases a disk or printer operation can be discontinued without locking up the computer by resetting the serial line. Sometimes this will also free a program that has locked up with the drive running. Similarly when a program crashes with the drive light flashing, you can reset the computer and then read the error channel or "m–r" around inside the drive memory for a possible clue as to what has happened.

1541 + 1702 = #?$*!! **Graham Reed Toronto, Ontario**

I'm sure you've all heard advice about not having your disk drive or cassette deck on the left side of your Commodore monitor. Well, if you want some real proof, with no fancy gear at all, here's what to do:

1) Get a telephone. One that has a separate base and handset is a must.
2) Turn on your computer and monitor.
3) Unplug your phone and take it to your computer.
4) Hold the base of the phone to the left side of the monitor, and listen to the handset.
5) Realize that the buzz you are hearing is caused by a magnetic field in the bell coils of your phone. . . and imagine what that may be doing to your disks.

Okay, now we know why you shouldn't keep your disk drive near the left side of your monitor. **But**, and this is a biggie, *if you are not having any problems with your disks and drive, do not change your setup!* The reason for this is very simple. All the disks you have formatted with your current setup (assuming that

that your drive is on the left. . .) have had this magnetic field affecting the drive. If you move the drive, then the field won't be affecting the drive, and your data will be fine, right?

Wrong. When the field is gone, the drive goes back into (or, from your point of view, out of) alignment. This means that your disks that were formatted on the left will have their data compensated for this magnetic field, and when the field is gone, it may have trouble reading the data.

I had very little trouble with my drive, except when trying to load some protected commercial programs. Tilting the drive on its side seemed to do the trick!

Do You Know Where Your Head Is?

Harold R. Skewes
Birmingham, Alabama

I program a little in machine language and once in a while my programs don't work out and the disk drive sticks or jams. Usually I take the drive apart and move the read/write head to solve the problem. It started to happen frequently, so I left the top of the drive's case off so that I could just reach in to move the head. My drive collected quite a lot of dust, and some other things, with the cover off.

Then I found a better way. One day as I reached to make the adjustment, the head vibrator protector that the drive was shipped with fell to the floor. As I picked it up, I took a good look at it, then turned off the drive and inserted it into the slot. The tab on the cardboard pushed the head back to track one, and the drive was ready to use again!

For yet another solution to head-correction, see the next bit.

More 1541 Tips

Wayne McDaniel
Terre Haute, IN

As one of the many owners of the Commodore 1541 disk drive/oven, I would like to offer a few tips:

First, it's been said before, but cutting a couple of pencils into about 2½ to 3 inch lengths and putting them in the screw holes increases the air circulation through the drive and keeps it cooler.

Second, I just had my drive realigned for the second time in six months, and the second shop where I had it done gave me some good advice. After the drive was properly aligned, they put some permanent epoxy on the drive stepping motor to prevent it from being knocked back out of alignment. They said that several people had taken their drives to other shops for alignment before bringing it to them, and the drive had just gone back out of alignment in 4–6 months. They told me that the epoxy seemed to be an almost permanent solution to the problem.

Third, the read/write head on my drive has on several occasions gotten hung up. I have found that if you take the disk out and tell the drive LOAD "*",8, this will get the head back into position.

Finally, I have to say that The Transactor is without a doubt the perfect magazine for all Commodore users.

That last part has been sent in by others already, but good advice is worth repeating – Ed

Break C64

Paul Bougard
Harmignies, Belgium

This program lets you put a virtual break anywhere in a BASIC program. Just set a break with the syntax:

sys 828,<line number>

For example, to stop your program at line 250, you would give this command in direct mode:

sys 828,250

Then, when you run your BASIC program, it will stop with the message "Break in line 250", when that line is about to be executed. The specified break only occurs once; if you run the program again, it will run as usual. This is a very handy debugging tool, and it saves you from having to put STOP commands into your program at various points.

Just run the loader program below to put "break c64" in place.

```
1 rem  break c64
2 rem  create a virtual break
3 rem  syntax:
4 rem   sys 828, line to break
5 rem  paul bougard 1986
10 i=828
20 read a: if a=256 then end
30 poke i,a: i=i+1: goto 20
828 data  32, 253, 174,  32, 107, 169, 165,  20
836 data 133, 251, 165,  21, 133, 252, 169,  85
844 data 141,   8,   3, 169,   3, 141,   9,   3
852 data  96, 165, 123, 201,   2, 240,  37, 165
860 data 252, 197,  58, 208,  31, 165, 251, 197
868 data  57, 208,  25, 169, 228, 141,   8,   3
876 data 169, 167, 141,   9,   3, 165, 122,  56
884 data 233,   4, 133, 122, 176,   2, 198, 123
892 data  56,  32,  52, 168,  76, 228, 167, 173
900 data   8,   3, 141, 104,   3, 141, 129,   3
908 data 173,   9,   3, 141, 109,   3, 141, 130
916 data   3,  96, 256
```

128 Notepad

John M. Paterson
Houston, Texas

Frequently when entering BASIC programs from magazines I add REM statements to identify the source of the article and the major functions and commands of the program. With machine language programs, however, REM statements are obviously not a possibility.

Now that I have a C128 I can record most of a 40-column screenful of instructions and other information by typing the screen exactly as I want to see it (except for the bottom 5 lines). Then I give the command:

```
bsave "filename.n",p1024 to p1824
```

This saves the screen in a binary file. Later, when I want to view the instructions I can place the cursor on the 6th line from the bottom of the 40-column screen and enter:

```
bload "filename.n"
```

using the same filename used to save it. This fills the screen with instructions without erasing the program in memory. The ".n" in the filename is used as a reminder that the file is a "notepad" file.

Save C-128 Variables in RAM Expansion

Richard D. Young
Greenwood, Nova Scotia

STASHing and FETCHing BASIC program variables to and from C-128 RAM expansion can be tricky. The information required for such an operation is included in the manual that comes with the RAM expansion module, but it must be interpreted with care. First of all, since I/O is enabled for the BASIC commands involved in RAM expansion activity, the top of variables pointer must be set to the beginning of the I/O memory area at $D000. Do this with POKE 58,208: CLR, then STASH and FETCH variable data from $0400 to $D000 of BANK 1 ONLY. Secondly, the C-128 BANK number used for DMA to and from RAM expansion is determined by the VIC RAM bank pointer at bit six of $D506. This bit must be set before activating RAM expansion activity associated with BANK 1 variables, and re-zeroed after such activity. Thirdly, and obscure characteristic of the C-128 bank configurations and their overlap dictates that the variable pointers that reside in $0000-$0400 always be written to RAM BANK 0. This means that variable pointer data must be read from and written to the C-128 in the BANK 0 or BANK 15 configurations, not BANK 1.

The following subroutines illustrate the process of saving C-128 variable data in RAM expansion banks.

```
100 : rem fetch variables from expansion ram bank 'rb'
110 :
120 poke 54534,(peek(54534) or 64): bank 1: slow
130 fetch 52224,1024,1024,rb: bank 15: poke 54534,4
    : fetch 10,49,49,rb
140 fetch 4,71,71,rb: return
150 :
160 : rem stash variables in expansion ram bank 'rb'
170 :
180 poke 54534,(peek(54534) or 64): bank 1: slow
190 stash 52224,1024,1024,rb: bank 15: poke 54534,4
    : stash 10,49,49,rb
200 stash 4,71,71,rb: return
```

128 BASIC Linefinder

Philip C. Herold
Seattle, Washington

If you've ever looked through the BASIC text storage area with a monitor's memory dump, trying to find that one elusive tokenized line, this will save you some eyestrain. It accepts a line number passed through the USR function, calls some routines in the BASIC 7.0 jump table, and returns with the address of the line. For instance, if you're looking for line 1000,

```
print hex$(usr(1000))
```

prints the starting address, in hex, of the tokenized line. The unused bytes in page 10 of the 128 provide a space that's just about right for the job.

```
A F0AC8  LDA #$D3    ;point usr vector at $0ad3
A F0ACA  LDX #$0A
A F0ACC  STA $1219
A F0ACF  STX $121A
A F0AD2  RTS
A F0AD3  JSR $AF0C ;float-to-integer
A F0AD6  JSR $AF8D ;search for line number
A F0AD9  BCC $0AEA ;branch if not found
A F0ADB  LDA $61
A F0ADD  LDX $62
A F0ADF  STA $65     ;int-to-float routine fetches from
A F0AE1  STX $64     ; these zero-page locations
A F0AE3  SEC         ; and requires this precondition
A F0AE4  LDX #$90    ; as well as this one
A F0AE6  JSR $AF0F ;integer-to-float
A F0AE9  RTS
A F0AEA  LDX #$11    ;"undef'd statement" error
A F0AEC  JMP ($0300)
```

If you use this area of memory to hold the code, SYS 2760 starts things running. You can put it anywhere else below BASIC text in bank 15 by changing where the USR vector points.

This could be handy if you want to use a monitor dump to slip some colour change or control characters into your REM statements. You could even use it from a running program as part of a scheme to make changes "on the fly"; for instance, to change REMs to PRINTs, or vice versa.

No-line LIST For the 128

K. van Mil
St. Ann's, Ontario

In The Transactor Volume 7 Issue 03, there was a simple method for the C64 to convert PAL-format assembler source code to CBM assembler format, called "Easy PAL to CBM Source Conversion". The POKE for convincing the 128 to do this was not given. On the 128, POKE 24,37 stops the printing of line numbers for a LIST. POKE 24,27 returns the system back to normal. The complete method looks like this:

```
open 1,8,2,"filename,s,w": cmd 1: poke 24,37: list
```

Then, to return things to normal after the LIST:

```
print#1: close1: poke 24,27
```

I like to use 128 mode to enter programs because the numeric keypad and the extra editing features makes program entry and correction a lot easier.

Table Look-Up Without Arrays

James MacFarlane
Islington, Ont.

Here are some quick ways to look up a string within a table without putting it all in an array.

```
10 print mid$("JanFebMarAprMayJunJulAugSepOctNov
   Dec", (mo-1)*3+1,3)
20 pa=1-pa: print "Pager is "+mid$("OFFON "
   ,3*pa+1,3)
30 dx=1-dx: print mid$(HALFFULL",dx*4+1,4)
   + " Duplex"
```

Line 10 will print the abbreviated name of a month given its number (1 through 12). Lines 20 and 30 show the use of two-state indicators that are toggled.

Mid-string functions save time and memory space since an array does not need to be defined or filled with data.

Here is the general form of the technique:

```
print mid$("FREDJOHNSUE  DAVE", (position-1)*
      (element length)+1,element length)
```

In this case the length would be four. The technique is easy to use and can be applied to a variety of programs.

Static Detector

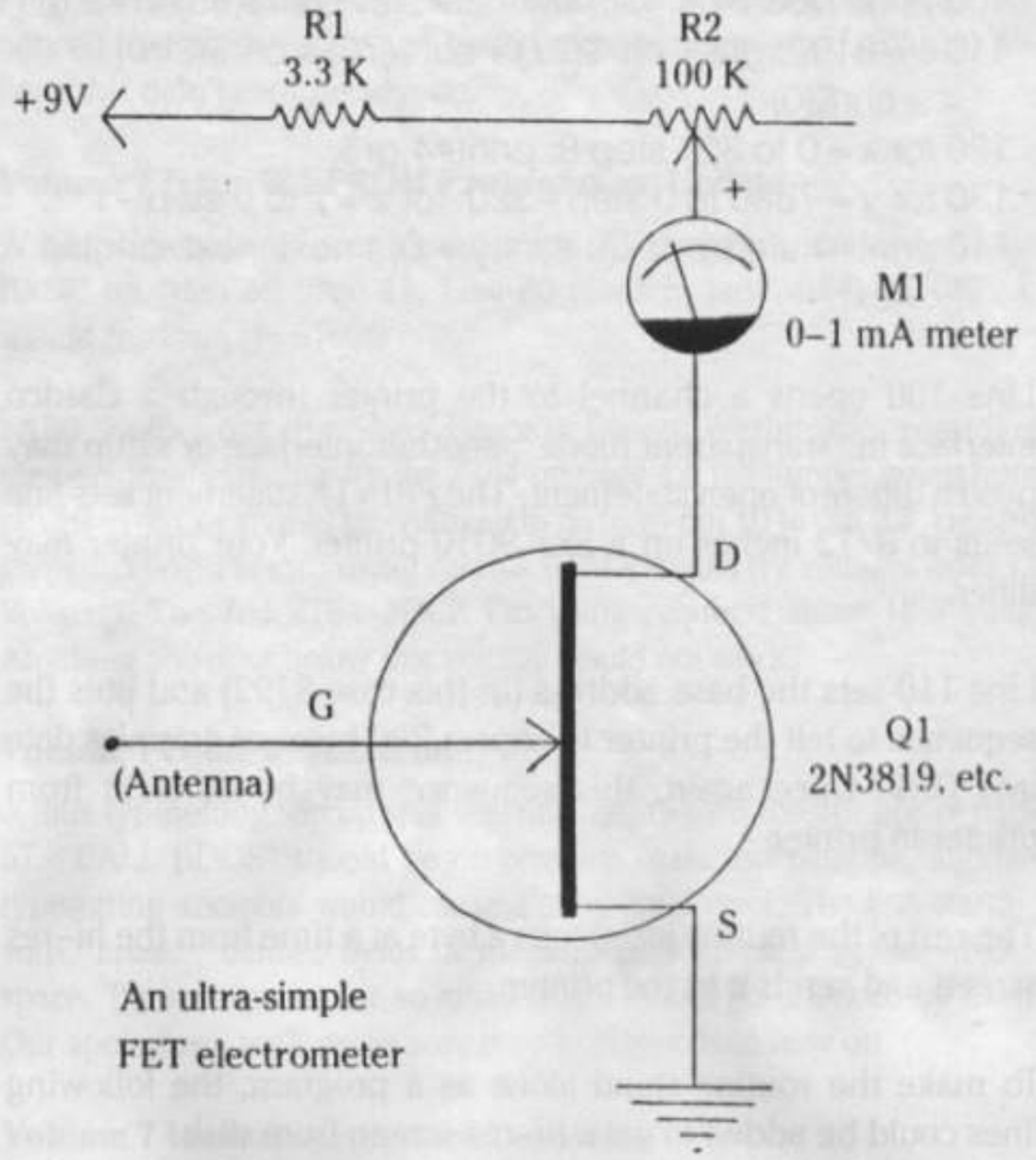
Andrew Fernandes
St. John's, Newfoundland

Recently a friend of mine just came back from the repair centre where he had his C64 fixed from static electricity damage. Too bad computers don't have Blue Cross. If the fellow's lucky, he might be able to pay back the loan within 20 to 30 years.

Not having a credit rating such as his, I decided to protect myself. The result, after a good one-hour search of eight years' worth of electronics magazines, is the following circuit taken from *Computer & Electronics*, January 1984 (Vol 22, #1), page 98 in a column written by Forrest M. Mims III.

This electrometer is quite sensitive and detects negative static charges from several feet away. Unfortunately, humans generally develop positive charges, so a simple solution is to replace the 2N3819 N-channel JFET with a 2N3820 P-channel device.

The circuit is simple enough to directly build into a small PLASTIC box, the antenna simply being a short piece of uninsulated wire. Use R2 to calibrate the circuit. A 9V battery will run the circuit for around 48 hours; when there is no danger to your equipment, you can shut it off by disconnecting the battery (you can install a switch to do this). Good luck!



An ultra-simple FET electrometer

Marble Madness Teamwork Tip

David A. Butcher
Cleveland, Tenessee

Want to increase your scores in that arcade hit, "Marble Madness" by Electronic Arts? Simply play *with* a friend! No, not *against* a friend, *with* a friend, as follows:

Some of you may have noticed that when playing in one-player mode, both joysticks/trackballs can control the ball. An annoying bug, right? WRONG. Put it to your use – BOTH of you can play the same ball, simultaneously! Be sure to have both joysticks or trackballs plugged in, and select single player mode.

Both of you can help control the ball, and best of all, if both of you use the "turbo" option (fire button), you add enough power to knock the steelie backwards in his tracks! And enough power to simply whizz by the vacuum nozzles without any deflection at all!

Scores of well over 24,000 points are easily attainable with this method, and the sixth frame is now easily reached.

Teamwork. It works!

Simple C64 Hi-Res Printer Dump

Mark Beckman
Pomona, California

I have seen quite a few routines in The Transactor and other magazines to dump a hi-res screen to a printer, but none of

them have been as fast, short and simple as this one. One word of warning: in order to be as simple as it is, this routine cheats – it prints the screen sideways.

```
100 print: open 4,4,4: print#4,chr$(27);chr$(65);chr$(8)
110 s=8192: gr$=chr$(27)+chr$(75)+chr$(200)
    +chr$(0)
120 for x=0 to 321 step 8: print#4,gr$;
130 for y=7680 to 0 step -320: for z=7 to 0 step -1
140 print#4,chr$(peek(s+x+y+z));: next: next: print#4
    : next
```

Line 100 opens a channel to the printer through a Cardco interface in "transparent mode"; another interface or setup may need a different open statement. The PRINT# statement sets line feeds to 8/72 inches on a Star SG10 printer. Your printer may differ.

Line 110 sets the base address (in this case 8192) and puts the sequence to tell the printer to expect 200 bytes of graphics data into GR$. Once again, this sequence may be different from printer to printer.

The rest of the routine just peeks a byte at a time from the hi–res screen and sends it to the printer.

To make the routine stand alone as a program, the following lines could be added to get a hi–res screen from disk.

```
60 if flag=1 goto 100
70 poke 51,0: poke 52,32: poke 56,32: clr: flag=1
80 input "filename";f$: if f$="" then end
90 load f$,8,1
```

Ribbon Alternative

Larry Cossaboon
Saint John, New Brunswick

While doing hi–res dumps on my 1526 printer, I found that I was going through printer ribbons more quickly than usual. To solve the problem, I took advantage of the printer's friction–feed capability and put a piece of carbon paper over the paper I was printing on and removed the ribbon.

Not only does the carbon paper save on ribbons, but it allows printing on clear acetate for use with overhead projectors, etc.

Formatting An Un–Notched Disk

A.J. Saveriano
Sparta, New Jersey

The following modification to the 1541 disk drive will allow you to format and write to an un–notched disk.

1. Carefully turn the drive upside down and remove the four screws.
2. Turn it right–side up (hold the drive together!) and remove the top.
3. Turn it on its side with the TOP to your RUGHT and remove the two screws that hold the metal shield in place. Remove the metal shield.
4. On the top left side of the PC board are five connector plugs in a row. The long centre one is the one we want.
5. The pins on this plug are numbered from back to front. We want numbers 12 and 13.
6. Pin number 12 will have an ORANGE wire and pin 13 will have a GREEN wire.
7. Install a jumper between these two wires and you will be able to format and write to an un–notched disk.

A jumper is a simple wire that joins or shorts two other wires or points on a PC board. You can install a SPST switch between these wires instead, then use the switch to easily go from normal to un–notched formatting.

Important: Make sure that the switch is in the normal (off) position before removing or inserting a disk, otherwise the DOS will not be aware of disk changes and it could get confused between disks and destroy data.

Protect Those Vectors!

Randy Rizun
Hamilton, Ont.

In the Bits and Pieces section of Volume 7 Issue 4, Philip Herold stated, "We all know what pressing RUN/STOP–RESTORE on the 64 does to our IRQ–driven wonders: it resets the IRQ vector and disables them.". Well. . .

I've found a way to preserve the IRQ vector, or any other vector, after a RUN/STOP–RESTORE. The main BASIC program loop is vectored through $0302, so by changing it, whenever the "READY" prompt appears, your vectors will be installed again. Here's one way to accomplish it:

```
entry    lda  #<setback  ;change the main loop vector
         sta  $0302
         lda  #>setback
         sta  $0303
         jsr  setirq
         rts

setback  jsr  setirq
         jmp  $a483

etirq    sei
         lda  #<irqrtn
         sta  $0314
         lda  #>irqrtn
         sta  $0315
         cli
         rts

irqrtn   . . .           ;irq–driven routine starts here
         . . .
         . . .
         jmp  $ea31      ;exit through end of irq routine
```

Sorting On The Fly

Martin Hofheinz
Stockton, CA

Sorting is certainly one of the most common things done by a computer.

The programmer has a wide variety of sorting algorithms from which to choose. Some are faster with pre–sorted lists; some are faster with un–sorted lists. Machine language sorts are the fastest, but they are tricky to incorporate into BASIC programs.

The usual approach is to first enter the data, then sort it after it is *all entered*.

Another way to sort is to enter the data, and sort each item *as it is entered*. This system works especially well with keyboard input, where data must be entered manually. The actual sorting is not too fast, but since it is being done at the same time you are looking up names, addresses, etc., the overall program run time can be shorter than if everything was sorted at once.

The following simple program illustrates a sample sorting routine for sorting lists of names and ages by either name or age. It creates pointers to the unsorted arrays "t$(" (containing the names) and "a$(" (the ages). The arrays of pointers are "a(" for the names and "b(" for the ages – these arrays hold the element numbers of the other arrays in sorted order. As each item is entered, it rises from the bottom of the list until it reaches its proper place.

The program is written as simply as possible, with no attempt at elegance.

```
10 input "how many names";n
20 dim a(n),b(n),a$(n),t$(n)
30 for j=1 to n: print: print "number";j
40 input "surname";s$
50 input "first name";f$
60 t$(j)=s$ + ", " + f$: rem combine surnames
   with first names
70 input "age";a$(j)
80 a(j)=j: h=j+1
90 h=h-1: if t$(j)<t$(a(h-1)) then a(h)=a(h-1)
   : goto 90: rem sort names
100 a(h)=j
110 b(j)=j: h=j+1
120 h=h-1: if a$(j)<a$(b(h-1)) then b(h)=b(h-1)
   : goto 120: rem sort ages
130 b(h)=j
140 next j
150 print: print "  sorted alphabetically:"
160 for j=1 to n
170 print t$(a(j)),a$(a(j))
180 next j
190 print: print "  sorted by age:"
200 for j=1 to n
210 print t$(b(j)),a$(b(j))
220 next j
```

TransBlooperz

Volume 8 Issue 1: Strange Cases of Backwards Braces

Programmers who typed in the C listing for "TrapSnapper" probably noticed that all the open and close braces were *reversed*. (Blush.) We hope this didn't confuse anyone.

Volume 7 Issue 6: EPROM Programmer Update

William Coleman of Green Cove Springs, FL wrote in with this fix to the BASIC program on page 41. Line 80 reads in part 'ad(4)=57087'. It should read 'ad(4)=5708**9**'.

..And another one from Alan Reece of Everett Washington, regarding the personality module for the 2764 on page 42: the jumper going from pin 10 to pin 24 should be changed to go from pin 10 to pin **22**. He also gives this tip: "Persons using surplus 2764's should try voltages from 12 Volts up. The AM 2764–2DCB I'm using required about 18.3 volts. Anything above or below this voltage would not work."

Volume 7 Issue 6: Textscan

A little typesetting slip up split one line into two on the 5th line of page 57. "CALL BDOS" should be on one line. Also, on page 58, another typesetting anomoly would cause a sumcheck error. The line starting with "linasc:" defines bytes in memory as 5 zeros, a colon, AND a space. Typeset spaces are so small that it would go unoticed by most. Our apologies – we'll make sure they're bigger from now on.

Volume 7 Issue 3: Keyboard Expander

John M. Paterson from Houston, Texas observed that Aubrey Stanley's "Keyboard Expander" was not fully compatible with C64 version 1 ROMs and sent the following fix. With this correction, the shift–F3 (clear to end of line) and logo–F3 (clear to end of screen) will work properly with the early machines.

Make the following changes to the program "ke.gen":

```
1000 rem program to create file "ke.1" on disk
1002 rem modified for 1st generation ROM
1004 rem by j. paterson, houston, tx
1006 rem 9/4/86
1007 rem changes in lines 1030,1040,1060,
       1070,1550,1900,4450,4470
1008 rem line 4475 added.
1030 for j=1 to 2753: read x: ch=ch+x: next
1040 if ch<>276696 then print "checksum error": end
1060 open 8,8,8,"0:ke.1,p,w"
1070 for j=1 to 2753: read x
1550 data 108,  27,   0,   0,  16, 129,  47,  10
1900 data 129,  32,   1,  10,  96, 136,  97,  48
4450 data  97, 162,   0, 129, 189,   6,  10,  98
4470 data  97, 144, 245,  96,  96,  97, 169,   1
4475 data  97, 145, 243,  96,  96, 112,  75,  69
```

Volume 6 Issue 6: VARPTR

This one's over a year old, but was only recently brought to our attention: Randy Winchester from Quincy, MA reported problems with the short VARPTR program on page 40. The problem is not with the program but with the instructions on how to use it to find the address of a string variable in memory. The text erroneously used a peek(v) instead of peek(v+2). The correct syntax is:

```
print peek(v+1)+256*peek(v+2)
```

LeTTeRS

Amiga coverage unjust: I feel that you are doing your loyal readers an injustice by including so many articles on the Commodore Amiga, especially in your May 1987 issue (volume 7, issue 6). I hope you realize that the Amiga has a completely different operating system than the other Commodore computers, and has a 16 rather than an 8 bit microprocessor. Commodore is now branching out into PC clones as well as the Amiga line.

I purchased a subscription to your wonderful magazine on the basis of the C64/C128 information and articles. If you intend to branch out into other areas, please cancel my subscription because I do not want a magazine for computers that I do not own. If I owned an Amiga, I would subscribe to a magazine like Amiga World. I hope you can realize the harm you are doing by publishing these articles, and that you will eventually lose your faithful following of dedicated Commodore 64/128 owners.

Bernard H. Weiss, Edison, New Jersey

Well, Bernard, we hate to disappoint readers, but let us just draw your attention to the front cover of any Transactor. Right under the name, you'll see a message prominently displayed: "The Tech/News Journal For Commodore Computers". Like the Amiga.

On the other hand, we admit you have a valid point. Our programs for the Amiga are not going to run on your 64 or 128, and the articles that tell you how it works aren't going to help you much either. And that's why, for the time being at least, we're restricting our Amiga coverage to something like 20 or 25 per cent of the magazine.

Sooner or later, though, something's got to give. We'll be very surprised if various models of the Amiga aren't around for a long time to come; whereas the 8-bit machines will probably dwindle in importance and market share. A magazine like ours eventually has to either adapt to the changing situation, or go out of business. You might say, "Well, why not start an Amiga version of Transactor, and keep the coverage separate?" Truth is, we'd love to, and maybe we will some day. Unfortunately, that's not a (financially) realistic option right now and, until it is, we're going to have to make shift with the 80 pages at our disposal.

Don't worry, though. We haven't forgotten our 64 and 128 readers, and we'll keep bringing you good things. Besides, are you SURE you wouldn't like to own an Amiga?

Ingratitude, publicly expressed: I am writing this letter in regard to the special issues of Transactor which you have been sending. These issues were started with Volume 7, Issue 2, and have continued to this day. Please don't misunderstand! I'm flattered by the trouble and expense you must go to in providing this unique edition. However, I believe that for the good of all your other readers, such expenses would be better made on increasing the number of excellent articles per issue (or, better yet, make The Transactor a monthly!)

As it is conceivable that you are publishing other "special" issues for other readers, I will attempt to identify those I am receiving. Their most recognizable feature seems to be in the omission of an Editorial Schedule for upcoming issues. Although this may seem a minor issue to some, the knowledge of major topics that are "Coming soon to your local Transactor" allows me to savour the anticipation for months in advance. It also proves of some help in deciding if an idea for a program and article will match with any planned issue topics. So please, guys, start sending me the regular issues of Transactor and save the specials for someone else!

Jack R. Farrah, Cincinnati, Ohio

So this is the thanks we get! All right, Jack R. Farrah, you've really got us steamed now! Anybody else out there want to get on the bandwagon? What about you, Milo Whistlebottom? Are you going to start whining because we print your Transactor without page numbers? And how about you, Henrietta Sloop? You want the 'toons like everybody else?

Okay, people, you've got it. No more special treatment, no more special issues. From now on, everybody gets the same Transactor

no matter who they are. And just to make sure you get the point,

(We interrupt this tirade to bring you a special announcement: as of next issue, Transactor is abandoning the practice of publishing theme issues, so please disregard the Editorial Schedules that have been appearing in all copies of the magazine apart from those being sent to Jack R. Farrah. Instead, we will be having one article (or a few) on a feature topic, with the balance drawn from the best other material we can lay our hands on. And unless we have a change of heart, that feature topic will NOT generally be announced far in advance, if at all. This new policy has come about mostly because we find ourselves delaying good articles issue after issue simply because they don't happen to fit the theme. And that isn't fair to the authors, or to you. So from now on, as a regular feature of Transactor, we will be omitting the Editorial Schedule. Look for its absence in future issues!)

Bang-bang floppy-copy: I just acquired a second 1541. Now I cannot find a fast copy program that does not bang the head on the destination drive (this includes Fast Hack'em 3.0, SuperKit and QuickCopy). Why do programs that work fine with one drive become destructive with two drives? Have they just copied each other's code? Can you suggest anything? I don't need sophisticated copying. Most of the time I duplicate my own or user group disks.

Warren Pollans, Davidson, North Carolina

As far as we know there's no real solution to your problem, Warren. The reason a copy program can avoid head-banging with a single drive is that it already knows where the drive head is when it starts writing to the destination disk. It knows that because the source disk was formatted, and the program was able to locate the head correctly according to the formatting information on the source disk. A destination disk in a second drive is an unknown quantity, however. Not only does the program not know where the head is at the outset of the copy, it does not have any means of finding out apart from the usual one of moving the head far enough towards the rim of the disk to bang against the stop. A partial solution would be to install one of the soft spring stops that are available as aftermarket upgrades for the 1541. The banging on your head won't stop, but the headache will.

Further, since most of your duplicating is done with disks that are NOT copy-protected, it's quite possible that some of the public domain disk copiers would be the answer. If you're transferring "file by file" to an already formatted disk, usually these programs won't bang the head on the destination drive.

Explaining the Drivelight Zone: The problem reported by Karl in the last issue (drive 0 on his 8050 was misbehaving because of a neighbouring monitor) is really quite common, and is due to flux from the flyback transformer interfering with the drive signals, which are incredibly weak at the beginning of amplification. Investigation of the monitor in question will probably show 'crazing' of the ferrite material used as the flyback core, due to repeated thermal cycles. One of these microscopic cracks has probably now formed a significant breach in the magnetic integrity of the core, with a significant increment of flux leakage.

Stated simply: (1) the TV signals screw up the drive, especially on the left side of the set, where virtually every TV manufacturer puts the flyback; and (2) prepare within a year or two at the outside to replace the flyback or the entire monitor.

Anthony J. Goceliak, Jersey City, New Jersey

Boat leaves Commodore stranded: Commodore missed the boat, again, with the new 1 Megabyte Amiga. The 68000 can run at 12 MHz; so why 7.13 MHz? Also, the 68020 can run at 14 MHz, 32 bit, which is about 235% faster than a 16 bit/12 MHz 68000, and about four times faster than Commodore's new Amiga. It is a shame to waste a megabyte of RAM on such slow speeds!

John R. Menke, Mt. Vernon, Illinois

Commodore isn't to blame for this one, John. While it's true that there is a 12 MHz (and more expensive) version of the 68000, and that the (much more expensive) 68020 can run faster still, the fundamental limitation in the 68000 micros is RAM speed, not processor speed. In fact, the Amiga gets you more speed for your money than other 68000 machines (like the 8 MHz Atari ST line) by running the 68000 on every second cycle of a 14 MHz clock, leaving the odd cycles free for use by the custom graphics and I/O chips.

Bird, plane, or Commodore 64?: I am writing in regard to an advertisement in the May issue of Computer Shopper on page 270. Swisscomp Inc. is advertising a 4 MHz 16-bit expansion card for the C-64 called "Turbo 64". According to the ad, it will increase the speed of the C-64 by 400%, "plug into the expansion port of the C64", contains a "16 Bit 65816 CPU with 64K of battery backed RAM", and "can address up to 16 MB of memory directly". The unit has a "special introductory price" of $189.95. Supposedly, they have a 1 MB expansion board under development.

Have you or any of your readers had any type of encounter with this product? I stumbled on the ad by accident, and thought it might be of interest. Just think of the possibilities with direct access of 16 megabytes of memory! (Isn't that more than the Amiga can address?)

Nolan Whitaker, Jeffersonville, Kentucky

Yup, the Amiga's stuck at a miserable 8.5 meg (9 on the new models). Of course, the Amiga has an Operating System that knows about that memory and can use it, which the 64 doesn't (unless Swisscomp is writing one, that is). Even so, it sounds like a great board, and we'd love to hear from anyone who can tell us more about it. And the price isn't bad, either.

Plus/4 Tech Info Source: In the July 1987 issue of Transactor (Volume 7, Issue 1), Jim Welch of Santa Clara, California, asked a question regarding the availability of schematic diagrams and other technical data for the Plus/4 computer.

Publications regarding the Plus/4 are few and far between, but I have found two good reference books for this machine. These are:

"Service Manual, Model PLUS 4 Computer", PN-314001-04. This is available from Commodore Direct Marketing, 1200 Wilson Avenue, West Chester, PA 19388. It costs $25 plus $3 S&H. The pinouts for the 6529 are given, but not the memory map. Complete schematics including the pinouts for all the external connectors are presented.

"Programmer's Reference Guide for the Commodore PLUS/4" by Cyndie Merten and Sarah Meyer. Published by Scott Foresman in 1986 (ISBN 0-673-18249-5). Order from local bookstore. This book has memory maps of the Plus/4, but no schematics or connector pinouts.

Anyone seeking to make serious use of the Plus/4 should strongly consider joining the Plus/4 SIG. Information regarding it can be obtained by writing:

Mr. Calvin Demmon, The PLUS/4 Users Group, Box 1001, Monterey, CA., 93942.

The group publishes a newsletter about eight times a year and has been instrumental in locating software suppliers for the Plus/4. The owners of orphan computers must stick together!

I now have a question of my own. I am interested in writing ML software which I want to store in the RAM below the BASIC ROM in the Plus/4. I understand how the RAM/ROM bank switching is done using locations $FF3F and $FF3E and can successfully load and execute code in the higher sections of RAM. My problem is: "How can I use the MLM in the Plus/4 to examine my code?" Once the ROM is switched out, the MLM is gone, so that it cannot be used any longer. I have tried every trick I can think of to try to download the MLM into RAM so that it is available when the ROM is switched out, but to no avail. I suspect that I am doing something wrong involving the I/O, but cannot figure out what it is.

Lee A. Cross, Dayton, Ohio

Thanks for the help, Lee. It's nice to know that Plus/4 owners have somewhere to turn for information.

According to the +4 manual under the "TEDMON" section, location $07F8 controls the memory source which TEDMON reads above location $8000 (ie 'M'emory dump, 'D'isassemble, etc.). If $7F8 is set to $00, TEDMON will look at ROM; if set to $80, it will read the RAM underneath.

Communicating Braces: I hope you can help me. I have to transmit via modem ASCII codes 123 ($7B) and 125 ($7D). These are the left and right curly brace characters respectively.

I am using a C-128 in C-64 mode to run Speedscript (v3.1). This stores screen codes in PRG files. I have a utility which converts screen codes to Commodore ASCII. My modem program in turn translates PETSCII to ASCII. My question is: how can I generate the braces? What screen codes do I type to start with?

Perhaps you have a simple answer, or a program which can selectively transmit the desired characters. I want to use a typesetting service which uses these characters as control codes for typesetting.

Joseph Francis, Zephyrhills, Florida

Most of the conversion from one set of character codes to another is done by manipulating the "zone bits" – bits 5, 6 and 7 – of each character to be converted. It happens that the ASCII codes for curly braces lie in the zone that also contains the lower case alphabetic characters (in binary, these codes have the form %011xxxxx). The Commodore screen codes for the lower case letters are in the range 1 to 26 (their codes have the form %000xxxxx). To get the ones you want, then, we can just convert 123 and 125 to binary (%01111011 and %01111101 respectively), alter the zone bits (we now get %00011011 and %00011101), convert back to decimal (27 and 29), and find the characters corresponding to these screen codes, which happen to be the left and right square brackets on the C-64 keyboard).

The only problem is, that's the wrong answer. Any reasonable PETSCII to ASCII converter will take into account the fact the square brackets have wandered out of their correct zones, and those screen codes will eventually translate to ASCII $5B and $5D, not $7B and $7D.

This raises the interesting question of what happens to screen codes like 91 and 93 ($5B and $5D) which would, like the upper case characters, come out as the same values in ASCII (the number 65, for instance, is both the screen code and the ASCII for upper case 'A'). Well, the screen codes in this range (actually $5B through $5F) are assigned to Commodore graphics characters that have no True ASCII counterpart, so it is likely that most PETSCII to ASCII converters simply leave them unchanged, including the one in your terminal program (the PETSCII codes do follow the screen code patterns rigorously, apart from the zone bits). A really smart converter, though, might take the PETSCII codes ($DB through $DF) for these graphics characters and swap them into the range $7B through $7F, which is what you want. If that were the case, which it probably isn't, the keyboard characters corresponding to the curly braces would be the shifted plus sign and shifted minus sign, respectively. Failing that, what you need is a terminal program that will send your characters untranslated; you would in this case do the conversion from PETSCII to ASCII (or directly from screen codes to ASCII) beforehand.

The most efficient way to write such a converter in BASIC is to use the table look-up method, in which you would have an array with 128 entries corresponding to the possible screen codes (assuming there are no reverse field characters in the Speedscript file – these use the screen code range from 128 to 255), each element of which contains the corresponding ASCII value. Given such an array, which you might call tr$, you would do the translation with a little program like this:

```
10 dim tr$(128): z$ = chr$(0)
20 gosub 1000: rem initialize the array
30 open 2,8,0,"0:ss-inputfile,p,r"
40 open 3,8,3,"0:asc-outputfile,s,w"
50 get#2,a$
60 print#3,tr$(asc(a$ + z$));
70 if st = 0 goto 50
80 close 3: close 2
90 end
```

This approach will also work if you choose to convert from PETSCII to ASCII instead of from screen codes to ASCII, only you'll need a larger array, as valid PETSCII values range up to 223.

Getting Poor Quick: I don't know how many of you folks got ideas about making "mucho big bucks" after you gained knowledge in writing machine language programs. We have heard or read about those who have made a fortune doing the same thing. No one writes about the many who failed.

The dream to make the program that will put us on Easy Street has overwhelmed some of us. I have learned much since that dream came into my head. I had visions of retiring from my electronic service business, and spending my leisure time writing programs for profit. After losing some money in advertising, I decided to do some market research. Boy, it is a tough world out there!

One source told me that there are over five thousand software development companies cranking out software, not to mention all the individuals. One of the biggest entertainment software publishers for Commodore, Apple, and Atari computers said that of the few programs they accept from many submissions, only one per cent does well in sales.

My attention turned to marketing my own software, so I checked into advertising. I don't mean a little classified ad either (lost money on that idea once). One source told me that the average reader response for advertising your wares in a computer magazine is from .1% to .5%. Far less than the one to two per cent I imagined.

Most of you probably write programs for other reasons. I did in the beginning. I did not plan to make money at first. However, the more I wrote programs, the more I thought about doing it for a living.

Making a "big hit" is not impossible, but it is very tough indeed.

John Augustine, Reading, Pennsylvania

Another IEEE Interface for the C–128: I am writing this letter in response to the inquiry about IEEE–488 interfaces for the C–128 in the letters section of the March 1987 issue. The Chemistry Department here recently acquired a Brain Boxes (25, Lynmouth Road, Algbruth, Liverpool, England) IEEE–488 Interface for connecting a C–128 to a spectrophotometer which uses the IEEE–488 bus. The interface can also be used with the C–64 and includes a superset of "wedge"–type commands in ROM. At time of purchase the cost of the interface was 77 pounds (including shipping and insurance). Brain Boxes accepts American Express, which simplifies currency exchange. I am enclosing further information, and hope this will be of benefit to your readers.

Tim Ballard, University of North Carolina at Greensboro

Thanks, Tim. From the info you included with your letter, the Brain Boxes interface is going to interest a lot of people. Anyone looking to use IEEE equipment with their C–128 or C–64 should check out the News BRK section in this issue for information from the Brain Boxes press release.

More Quicksilver IEEE Info Wanted: I would like more information on the Quicksilver IEEE interface described by John A. Spencer (Letters, Volume 7 Issue 6). I have the 128 and an old MSD IEEE interface that works in 64 mode only. It requires a SYS call to get from IEEE to serial and another to get back. A friend here in Jackson has the BusCard (on his 64) and it seems to handle devices on both buses much more elegantly. How about the Quicksilver? Can it 'get' from the IEEE and 'put' to the serial transparently? Will the Quicksilver work in CP/M mode? Could I use my IEEE and serial devices both in CP/M mode?

In the same letters column, Doug Hurd of British Columbia was looking for information on sources for long cables and other unusual parts, plus an inexpensive modem. For the long cords, he might try Precision Peripherals and Software, P.O. Box 20395, Portland, Oregon 97220, phone 503–254–7855. A source for other odd parts is Black Box Corp., P.O. Box 12800, Pittsburg, Pennsylvania 15241. Their order number is 412–746–5530; their tech support line is 412–746–5565. As for a modem, Doug might want to check out the Total Telecommunications auto–answer, auto–dial modem from C.O.M.B. for $19.00. If he can live with 300 baud it is great. I have been using mine for a year now on my BBS, and also call out on it from time to time. I have seen ads for the Commodore 1600 from them at the same price, so make sure they know which one you want when you're ordering. C.O.M.B.'s address is 14605 28th Avenue N., Minneapolis, Minnesota 55441–3397. They have a toll–free line for U.S. residents at 1–800–328–0609.

Rick Crone, Jackson, Tennessee

Our apologies, but not owning a Quicksilver interface makes it difficult to answer your questions, Rick. We also do not own a single BusCard, and the fact that Batteries Included has all but joined the ranks of Info Magazine's R.I.P. column, you might find it difficult to get one. Electronic Arts has acquired rights to some of their products, so it's possible they might have some information for you.

On the other hand we do have some good news, however self-serving it may seem. The G-Link will make IEEE/serial transfers, and Richard Evers has modified Jim Butterfield's "Copy-All" to take advantage. It will appear on every Transactor Disk starting with number 19 for this issue. Up until about 6 weeks ago we thought there would be no more G-Links. The parts for them are quite common, but making the printed circuit boards is a fairly involved process. Then, while cleaning out the warehouse in preparation for our move from Milton to Richmond hill, we found about 200 more unpopulated G-Link boards. So, based on this discovery and the number of requests we've had for them (not to mention the number of orders we've had to send back because we thought we didn't have any left), we've decided to continue offering them. See News BRK for more details.

Super–C 3.0 Fix Available: I have just finished reading the March Transactor, and would like to make some comments concerning the articles I've read. The first comment I have is that there is a bug in Super–C 3.0 that has been fixed with the 3.02 version. The new version fixes the 'getc' problem when using an RS232 file – the old version would hang up the system when it got to the first 'getc' for the file. Present owners may want to ask Abacus for details. Enclosed you will find a letter I got (from Germany!) that confirms the problem.

The brand–new version of Pocket Writer 2.0 has a funny bug in it. If you boot the disk from a cold start, i.e. put the disk in the drive and turn the computer on, the disk will make horrible sounds! However, if the disk is booted from a warm start, i.e. press the reset button, the disk doesn't knock any more. By the way, the new features of Pocket Writer 2.0 are great – including mouse support, RAM disk, and spelling checker.

Mark Bonnema, South Holland, Illinois

Mark certainly can't complain about the response he got on his Super–C question. The "letter from Germany" he received was from Franz Hauck, one of the two authors of the compiler. Hauck not only confirmed the RS232 bug, but also went out of his way to correct a couple of minor errors in the C source code Mark sent him, and returned Mark's disk with the corrected code. Now that's customer service!

TeleColumn

Transactor Area on CompuServe

Almost 2 megabytes of Transactor articles have now been prepared for uploading to CompuServe. Yes, this means that the much touted Display Area is still not operational, but man what a job! We figure there's about 3 more megabytes of text to go before almost every Transactor article ever printed since Volume 4 Issue 01 will be available online.

Some of the articles were not going to be uploaded due to diagrams. But many of them have been "hand-translated" to low-res replicas. They may not be as good as the printed versions, but they get the point across enough that the article didn't have to be excluded.

Also in the uploading queue is The Transactor Writer's Guide.

The Transactor Data Library currently has every program ever published from Volume 4 Issue 01 to Volume 5 Issue 06. As mentioned last issue, we wanted to get the programs up first so the articles could refer to them by name. There's about 12 disks to go and we average about 1 per night. After this issue is done we'll resume the uploading activity and with any luck we'll have our entire history for the past 5 years available online by next issue (tongue firmly between teeth).

New Rates on CompuServe

From April 1 to May 30, 1987, CompuServe announced that their prime time charges would be the same as their non-prime or night time charges. Now, however, CompuServe has decided to make the daytime rate reduction permanent. 300/450 BPS is now $6.00 per hour and 1200 BPS is $12.50 per hour. 2400 BPS rates are the same as 1200. TeleColumn suspects now that daytime rates have been reduced, the nightly charges will also come down, at least for 1200 BPS if not 300/450. Watch this spot next issue.

Time Saving Tips

On CompuServe, as on any online service, time is money. Here are a couple tips that may save you a little or a lot.

Even at 300 BPS it's possible for CompuServe to send you messages faster than you can read them. Agreed, many can read faster than 30 characters per second, but messages are not always grammatically perfect. So you might stop to read a sentence or paragraph twice, but the system just keeps sending. Even if you stop the flow with a Control A, the clock keeps ticking. One alternative is to capture the messages and read them off line.

If your terminal program has capture buffer capabilities, sign on and open the buffer at the main "Function:" prompt. Enter RTN (Read Threads New to you), RF (Read Forward), or whatever you use most. Following the first message (which will now be in your capture buffer) you'll get the "Read Action" prompt. Enter NS at this point for "No Stop". Messages will be captured non-stop from where you start through to the last, or until your buffer is full, whichever comes first. Now save the buffer to disk and use your favourite file reader, text editor, or word processor to read the messages at your convenience.

To reply to a message off line, use your word processor and enter "RE" followed by the message number you want to reply to. Type the rest of your reply making sure there is a carriage return after every 80 characters (I usually type the whole message and put the CRs in later). Now load your capture buffer with the reply. Sign on, navigate to the main "Function:" prompt, and dump the buffer contents. "RE 7500" means REply to message 7500. At this point CompuServe usually prints "Enter You Message", but your terminal program won't wait for it. The text will just continue pumping into the system, but that's ok. When done, use CTRL Z or /EX, depending which editor you have selected. "/EX" is for exiting the FILGE editor and could be included as part of your off line composition if you like, followed by "S" for "Send" on the next line. You may want to use PREview before Send the first few times to be sure you have it right.

Once you have a little practice, all your replies can be capture dumped at full bore. You only need be sure that each reply is separated by the correct commands. The number of replies you can pump in non-stop has not been tested, but 3 or 4 shouldn't be a problem.

If this technique becomes routine for you, consider 1200 baud. It's twice as much as 300/450, but it's four times faster and potentially less expensive for uploading off line replies.

RLE Notes

Two issues ago, Christopher Dunn explained the concept of RLE (Run Length Encoded) files; a method for displaying high resolution pictures on CompuServe. Chris is also the author of two conversion programs – RLE2HR converts RLE files to high-res files for the 64, and HR2RLE converts high-res information at $2000 in the 64 to RLE format so it can be uploaded to CompuServe. The programs were supposed to be on Transactor Disk #17, but didn't quite make it. Both programs and the documentation files to go with them are now on Transactor Disk #18 AND #17. Our apologies for any inconvenience.

The article explained everything about RLE files, but one point may have been left unclear. RLEs begin with ESC GH. The "GH" stands for "Graphics Hires". In hex, the values are $1B for ESC, and $47,$48 for GH. These are the true ASCII values for G and H, but in PETSCII the values for capital G and H are quite different, which may have lead some to generate the wrong values in the RLE 3 byte header.

To summarize again briefly, the header is followed by bytes that are always in the range from decimal 32 to 127. These represent pairs of on and off pixels in the screen area which is 256 pixels across by 192 down. Decimal 32 is added to each byte to keep it in the range away from Control type characters. Therefore, following the 3 byte header, the sequence of decimal 40, 52, 78, 35, 127, 32, 55, 70, would mean 8 pixels on, 20 off, 46 on, 3 off, 95 on, 0 off, 23 on, 38 off.

Theoretically, you would continue the pairs until all 256×192 (49152) pixels are defined. However, if a lot of pixels near the end were all off, the file end marker could be thrown in early. This is ESC GN for "Graphic Normal". In hex, the values are $1B, $47, and $4E.

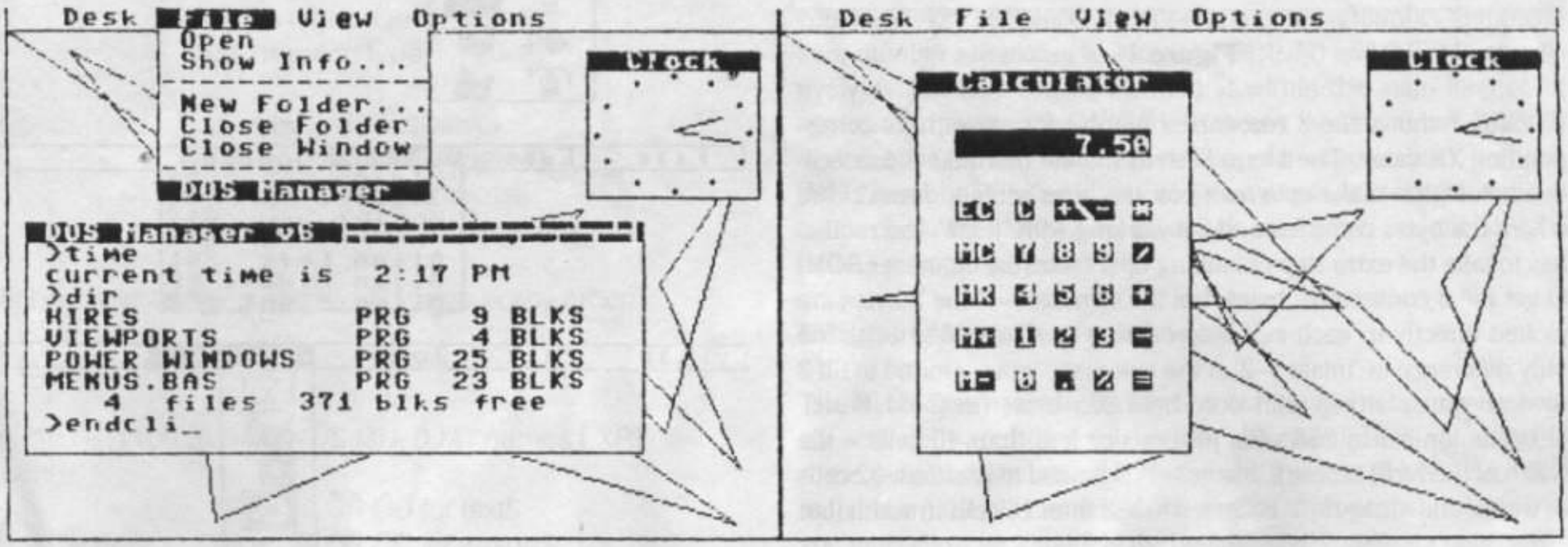
Letters to TeleColumn

F.J. Warner of Vancouver, BC, wonders why there has been no mention of "The Punter Network" in TeleColumn. Actually, we've been wondering why we haven't received just such an article. Much of the material we get is on topics well past the "worn out" stage. An article on PunterNet would really catch our attention *(hint, hint)*.

Mouse Driven Menus

Anthony Bryant, Winnipeg,Manitoba

. . .a hires Menu System for the C–64 and C–1350 mouse. . .



Two sample screens showing off 'Menus', 'Hires', and 'Viewports'.
The little clock is updated every minute with simple multi-tasking in BASIC.

Pull–down Menus

The newest Microsoft BASICs (such as Amiga Basic) have built–in commands to create and manage a pull–down Menu System. If you've never used pull–down menus, you might not appreciate their advantage in offering both the ability to browse through a list of menu commands and the ability to execute them, all in one visually–oriented package. At the same time, the 'look' of the menu is open, allowing creative design. The welcome introduction of the C–1350 mouse prompted me to work out this concept on the C–64.

Two new commands, MENUS and MOUSE presented here, manage a system of menus on the hires screen. If you wish, some or all of your menu names or items can be graphic images (pictures, icons, glyphs) instead of text. The Menu System holds a maximum of 56 items and 7 menu names in the menu bar.

MENUS is designed to co–exist with 'HIRES' and 'VIEWPORTS' (see the Transactor Vol.5 Iss.6 and Vol.7 Iss.2) drawing on the same hires screen (at $E000) but is self–contained (uses no routines from either) so that, if the user selects a menu, any drawing on the hires screen is frozen, while the Menu System goes into action, and unfrozen continuing drawing where it left off. Menu names and items can be disabled (dimmed or ghosted) or items can be marked (with a checkmark). An alternate command–key sequence to access menu items is also supported.

MOUSE sets up a sprite cursor, and manages menu selection as a background task (interrupt driven). MOUSE requires the use of the defacto Microsoft/Amiga standard left and right buttons. (See 'A Two Button Mouse' Transactor Vol. Iss.) The left button is called the 'Select' button and the right button is the 'Menu' button. To activate the Menu System, the user presses the menu button (moving the mouse cursor over a menu name within the menu bar) and while still keeping it pressed, moves the mouse cursor in the list of menu items that appear below the menu name. To select an item, release the menu button while over a highlighted item. The selected item will flash, reinforcing the selection. If the user wishes not to select anything, he simply mouses out of the menu box. It's all . . . well . . . intuitive.

'MENUS.BAS' is a BASIC loader that, when run will create a program file called 'MENUS' on your disk. (Make it the same disk that holds 'HIRES' and 'VIEWPORTS').

'MENUS DEMO' is a Basic program that demonstrates the new commands and encourages user experimentation.

The syntax of the two new commands is quite varied, but first let's look at some definitions for the Menu System.

Graphic Forms

The hires screen is object–oriented. That is, it doesn't know text from images – only bitmap objects. MENUS recognizes three different types of graphic forms. All of the menu names and items use string variables (because its the most compact method of storing a sequence of bytes). In order to differentiate the 3 graphic forms, an extra variable, XR is used with a menu name$ or item$.

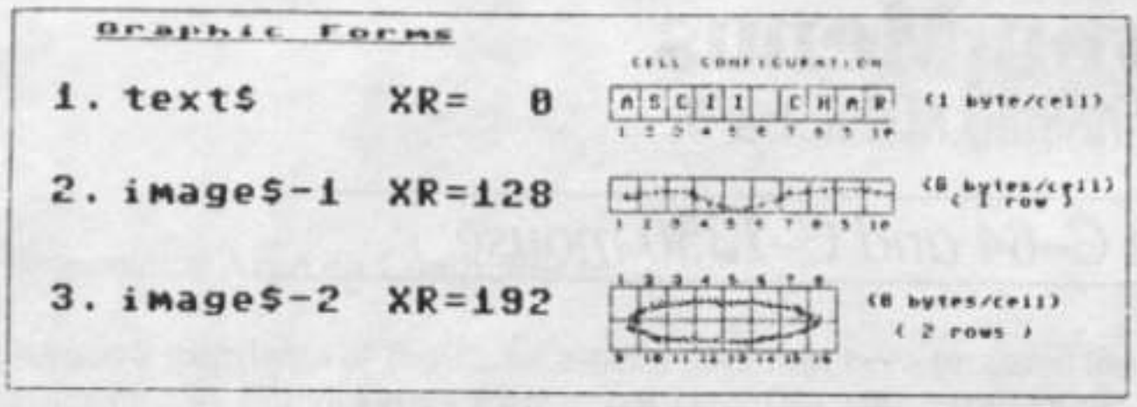


Figure 1

FIGURE 1 shows the 3 recognized graphic forms, with its corresponding XR value. The Menu System routine that puts 8 consecutive bytes (that make up a 'cell') on the hires screen, doesn't care where the bytes come from. If it's working with 'text$', the routine has to take the extra step of looking up a table (the character ROM) to get the 8 consecutive bytes, but if it's 'image$–1' the 8 bytes are plotted directly in each cell before going on to the next cell. The only difference in 'image$–2' is the automatic wrap–around to fill 2 rows. Because strings can only hold 255 bytes (in C–64 Basic), 'text$' is limited to 255 cells (in practice less than 40 cells – the width of the hires screen), 'image$–1' is limited to less than 32 cells in width, and 'image$–2' is limited to less than 16 cells in width (but 2 rows).

Offset Columns

In the extra variable, XR, it's the two most significant bits, bit7 and bit6, that determine which it is of the 3 graphic forms. The remainder of the bits in XR can be used to move the column position (offset from the left edge). These bits are collectively called OC and if omitted the Menu System handles positioning automatically. OC allows menu names to be spread evenly across the menu bar (or bunched up – your choice). It also allows grouping of 'image$–2' menu items in a menu box.

Key Codes

Alternately, the remainder of the bits in byte XR, (bit5–bit0), can be used to store an appropriate command–key sequence code. These bits are collectively called KC, and are the keycode values found in location $C5. For example A=10, Q=62, P=41. (The 'Inner Space Anthology' does not list these codes, but COMPUTE! has published them several times and Raeto Collin West's PROGRAMMING THE 64 has a nice table of keycodes on page 161). Once added to the menu item (where together with a stylized Commodore logo they are printed in the menu box) the user can make menu selections with the keyboard instead of the mouse by pressing Commodore–key and that key, together.

Menu Types

There are instances where you would want to have menu items selected, but not acted upon. All that happens is a checkmark is placed next to that item. And there are occasions where you would want to disable (keep from being selected) certain items or even menu names. A disabled item or name is dimmed (faint or ghosted) – still readable but less distinct. A value of (0–2) for MT determines this. Enabling or disabling a menu name or item before or during your program is an easy procedure.

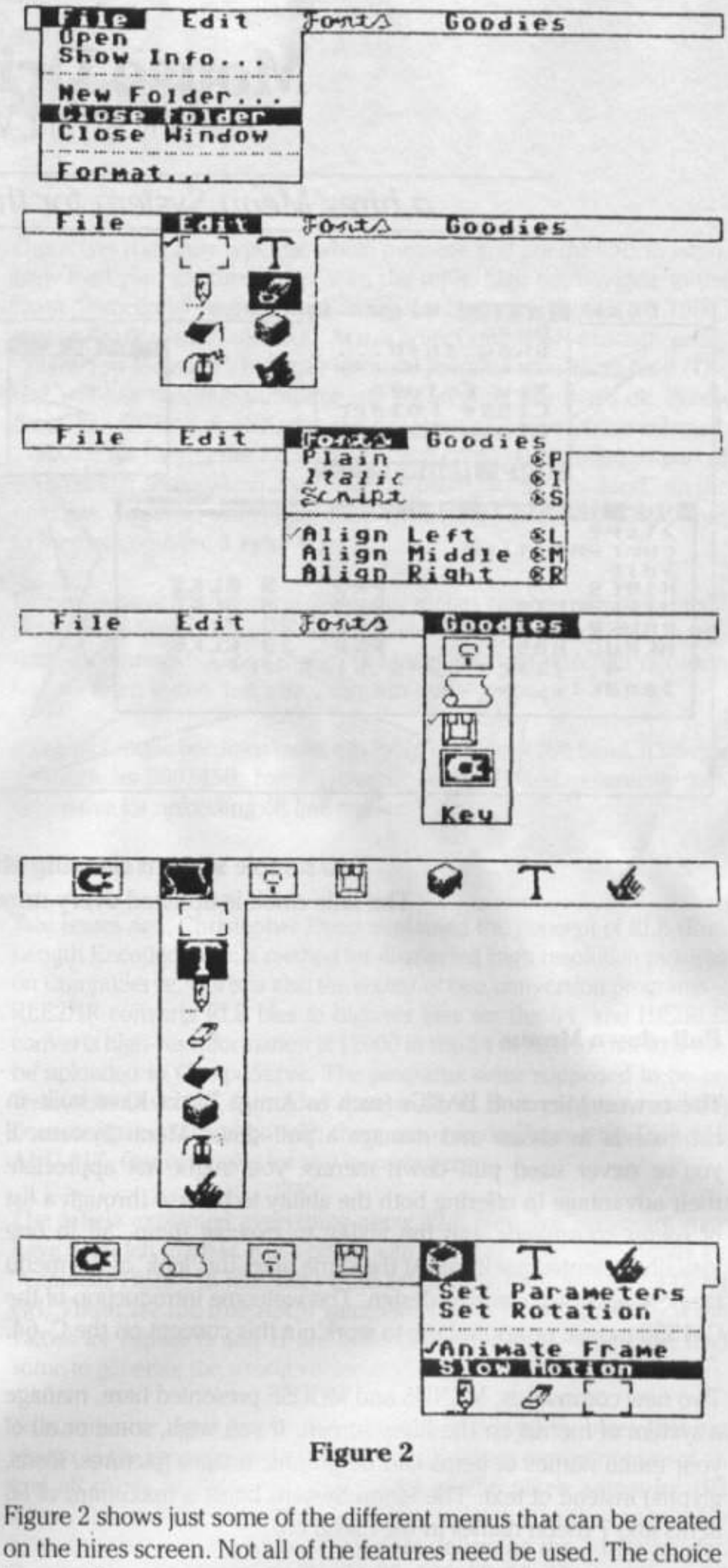


Figure 2

Figure 2 shows just some of the different menus that can be created on the hires screen. Not all of the features need be used. The choice is up to the programmer in the design of the 'look' of his menu and in the requirements of the application. But the Menu Sytem has a number of default conditions, that speed up building a menu. Let's look at the syntax and those defaults now.

A LA CARTE (. . .the main course. . .)

The commands are listed in the order in which they would normally appear in an application program.

To initialize the system . . .

1000 SYS MENUS CLR [,BG,FG]

where: BG is menu background color (0–15)
FG is menu foreground color (0–15)

clears out any previous menu lists, and initializes the system, with optional color parameters.

Next, comes a series of names and items to fill the menu list.

where: MN is menu number (1–7)
MI is menu item (0–8)
MT is menu type (0–2)

types: (mutually exclusive)
- MT=0 for disabled (dimmed)
MT=1 for enabled (normal)
MT=2 for marked (checkmark)

Submit a list of menu names and items in any order.

To add a menu name. . .

```
1100 SYS MENUS,MN, 0,MT,name$ [,XR]
```

where:

(MI=0) XR= 0 OR OC for text$
XR=128 OR OC for image$–1
XR=192 OR OC for image$–2

'OR' is the logic OR, effectively adding the OC bits to XR. OC is the offset column position relative to the left edge and can have a value of (1–40). The default for XR is 0 (text$ and no special offset column).

To add a menu item . . .

```
1200 SYS MENUS,MN,MI,MT,item$ [,XR]
```

where:

(MI<>0) XR= 0 OR KC for text$
XR=128 OR KC for image$–1
XR=192 OR OC for image$–2

'OR' is the logic OR, effectively adding the KC bits to XR. KC is the key code. You can't have a KC with image$–2 form, since an OC is needed if image$–2 forms are to be grouped. The default for XR is 0 (text$ and no key code).

To display the menu bar . . .

```
1300 SYS MENUS ON [,BG,FG]
```

where:

BG is menu background color (0–15)
FG is menu foreground color (0–15)

the hires screen area under the menu bar is saved before the menu bar is printed.

To change menu type . . .

```
1400 SYS MENUS,MN,MI,MT
```

within the course of a program

To remove the menu bar . . .

```
1500 SYS MENUS [OFF]
```

the hires screen area under the menu bar is restored.

Menu Selection Messages

When a menu item is selected, either by the mouse or alternate command–key sequence, the appropriate menu number and menu item number are placed in locations (MNUM) and (MITM), respectively, where they can be PEEKed at within the main loop of an application program. (MNUM)=0 means no menu or menu is disabled and (MITM)=0 means no item or item is disabled. Location (MFLG) can be PEEKed if your program needs to know if the menu bar is displayed. (MFLG)=0 means no menu bar. See the MENUS DEMO program for examples.

Designing Menu Text and Images

The Menu System automatically lays out the menu box (every time it's activated) according to the lists submitted, and according to some general design rules. For example, it places (if MT=2 for that menu item) the checkmark in the first column of the item row, so you must allow for a space at the beginning of your text$ or image$–1 if you use the checkmark. image$–2 form requires an OC value for its placement and the checkmark (if used) is placed to its left. Incremental OC values allow image$–2 forms to be grouped along the same row. For text$ the default case is the lowercase ROM set. To switch to the graphic ROM set, concat text$ to CHR$(141). CHR$(14) reverts to lowercase. You can make use of the graphic ROM set in a text$ for divider lines, or even crude graphics. Use the MENUS DEMO program to experiment!

Designing Images is made relatively straight forward using a good font editor. I used Charles Brannon's excellent ULTRAFONT+ because it readily outputs the DATA statements (which you can then use to build a string). Just clear out sufficient cells and treat it like a bit–map editor (you can use the mouse). image$–1 (1 row form) can be used to depict different fonts, for example. image$–2 (2 row form) can be used for a larger bit–map area. The cell width of your image designs will determine the general layout of the menu box. The MENUS DEMO program contains a few examples.

MOUSE GLACE (. . .and for dessert. . .)

The MOUSE command sets up the interrupt driver routine, which polls the mouse position, moves the sprite, monitors the left and right buttons and handles all the details of displaying menus.

Let's look at the variety of syntax.

To change the cursor . . .

```
2000 SYS MOUSE,cursor$
```

allows the design of a new cursor shape (See 'Designing a Cursor' below.)

```
2000 SYS MOUSE,0
```

sets the cursor to transparent

```
2000 SYS MOUSE,1
```

sets the cursor to the built–in arrow shape

```
2000 SYS MOUSE,2
```

sets the cursor to the built–in cross shape

To show the mouse . . .

```
2100 SYS MOUSE ON [,COLOR]
```

activates the mouse interrupt driver with optional color parameter (0–15).

To hide the mouse . . .

```
2200 SYS MOUSE [OFF]
```

removes the mouse interrupt driver. Use to turn the sprite off, in order to access serial devices (disk, printer, etc).

Mouse Messages (Mouse Droppings?!)

While the mouse is ON, it continually updates several locations which an application program can PEEK. Location (MB)=0 if no button was pressed,(MB)=1 if right button was pressed,(MB)=255 if left button was pressed. Locations (MX) and (MY) can be PEEKed to get the pixel positions for 'HIRES' to use to draw on the screen. Location (MX+1) is the x–high bit. Location (MY+1) is equal to 199–MY for use in a non–Cartesian system. The MENUS DEMO shows more on this.

Designing a Cursor

Since the cursor is a sprite (sprite0 in fact), you can use your favourite sprite editor to design with. I used Charles Brannon's SPRITE MAGIC editor, because it generates DATA statements (which can then be built into a string). The file "ARROW.SPR" on the distribution disk, contains the data for the built–in arrow sprite, which you can load into a sprite editor to use as a template for your own design. Of particular importance is the hot–spot position (tip of the arrow).

About MENUS DEMO program

This program, after building the menus, just executes a running loop, polling the menu and mouse variable locations. It just prints the selected (by mouse or alt–command–key–sequence) menu and item numbers on the old text screen. Experiment with this program. Exchange menu names and items, move graphic images about using various OC values. Change KC values. Figure 2 shows some of the configurations possible. Change the action of, say, the right menu button, so it toggles the menu bar on/off. Try the random SYS DRAW and note what effect going to the menu has.

It doesn't take long to get used to creating and using pull–down MENUS, and before long you'll find yourself with a desire to change some design criteria of the Menu System. (Everybody's a critic).

Inside Mouse Driven Menus: Some Technical Notes

Assembling the Components

The PAL source code is quite long, but is basically a library full of small modules, software components, that when assembled work as a system. You can't do Menus without doing Windows and you can't do Windows without doing Viewports. Modularizing all these routines was the only way to go (to remain sane). The modules (refer to the PAL source file) are arranged in the following catagories:

(Editor's Note: Anthony's source is a whopping 1000 lines! It couldn't possibly be printed here – refer to Transactor Disk #19.)

1. Menu support routines
2. Window routines
3. Viewport routines
4. Plotting routines
5. Mouse support routines
6. Sprite routines

'HIRES' operates in Bank 3, using the hires screen at BMAP1 = $E000 and color memory at CMEM1 = $CC00. So does 'MENUS'.

1. Menu support routines

A menu is mostly 'Lists' – lists of names and items and types. String variables were chosen to store these 'lists' because it's the most compact method of storing a sequence of bytes. The Menu System recognizes three different 'graphic forms' (determined by an extra byte, XR, following the string).

form	XR
text$	00xxxxxx
image$–1	10xxxxxx
image$–2	11xxxxxx

Since each string can be totally defined with just three bytes (length and low byte/high byte pointers to the start of the string) only four bytes are needed to define each menu name$ and menu item$. To hold 56 item$ and 7 name$ requires just 252 bytes i.e a page of memory – called MLIST. The routines SETLIST and GETLIST store and retrieve the four bytes to/from respective 'slots' in the page of MLIST. As a consequence, menu names and items may be submitted to the menu list in any order with the SETMTOP routine keeping track of the maximum no. of menu names and items in each menu.

Also, an area of memory is set aside for the MENU DATA STRUCTURES. MTOP holds the max. no. of menu names (#MN=0) and items in each menu. MNLC, MNTR, MNWD and MNDP hold the dimensions of the menu bar (#MN=0) and each menu box. #MT defines an item's type as disabled (dimmed), enabled (normal) or marked (checkmark). SETTYPE and GETTYPE store and retrieve MTYP, which for each item is stored in MDIM and MCHK (for #MN=0 the type of each menu name – disabled or enabled – is stored in MDIM). The MDIM or MCHK byte holds the type for 8 items (one bit in the byte for each item).

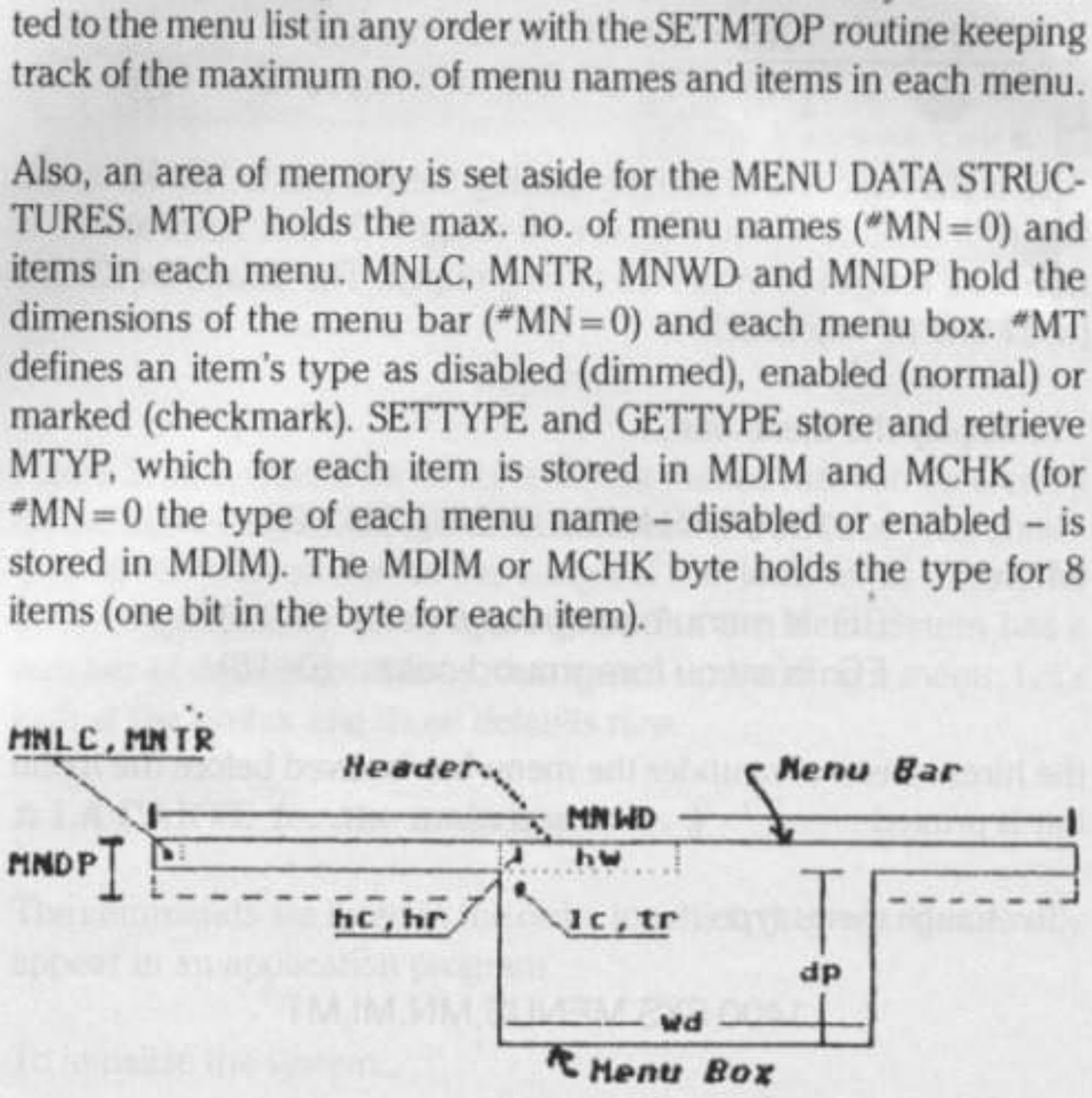


Figure A

Figure A shows the dimension variables for the Menu System. MNLC and MNTR (#MN=0) determine the top-left corner of the Menu Bar when it's attached to the hires screen. MNWD determines the width. These three variables are preset, but could be altered to suit.

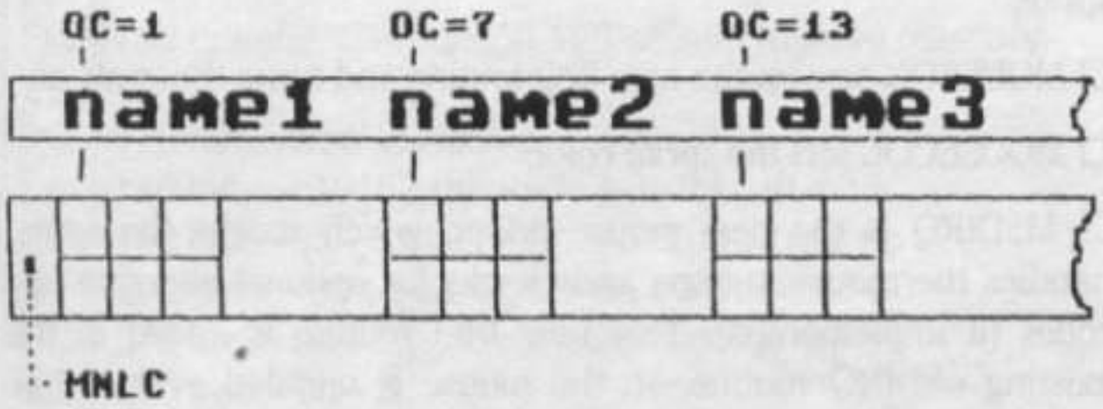


Figure B: General Menu Bar Layout

Figure B shows the positioning of menu names by OC, the offset column.

☐ MENUSIZ routine just alters MNDP, the depth of the menu bar to accommodate 'image$-2' graphic forms.

☐ MENUBAR prints the menu bar on the hires screen and handles automatic column positioning if no specific position is given.

☐ MENUSEL is called by the mouse driver (by MENUBTN) whenever the menu button is pressed. It checks if the mouse is within the menubar area – if not returns with carry set – then scans each menu name for horizontal position, to check if the mouse is within a menu name's 'area of influence' (determined by the length of the name$). You can fool MENUSEL by submitting menu names with wrong order column positions –it just ignores the out of order name! MENUSEL highlights the menu name and returns to the mouse driver.

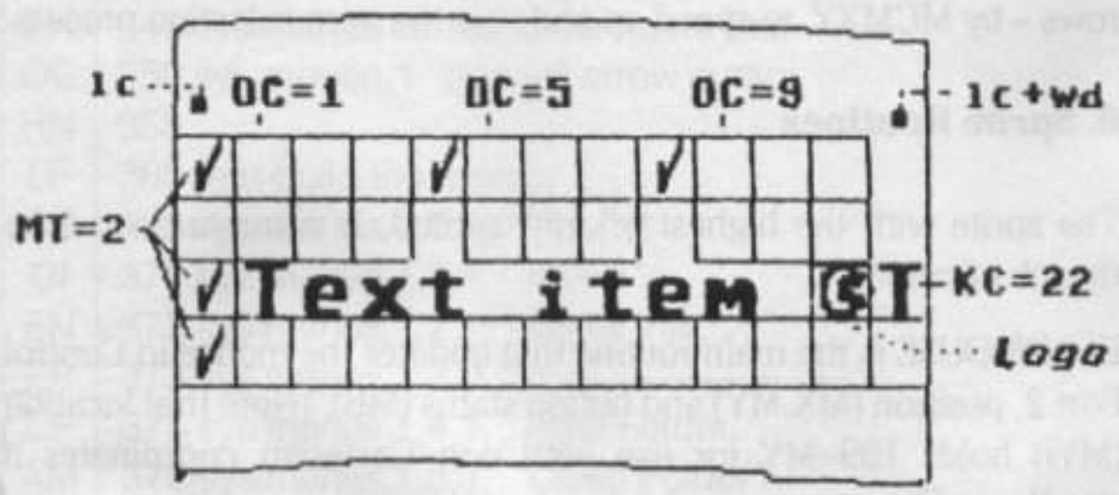


Figure C: General Menu Box Layout

☐ MENUBOX is also called by the mouse driver (by MENUDOWN). It's job is to look at the SELected menu's items to determine the dimensions of the menu box. (You can change the list of items or re-submit new ones between accessing the Menu System). It goes through the list of items keeping track of each items' length (and group 'image$-2' total length) and stores the dimensions in the MENU DATA STRUCTURE. The GETLIST routine is invaluable (in this and the MENUPLT routine) in simplifying this job. GETLEN always presents the correct cell length to enable placement and sizing. Thus the menu box is automatically sized.

☐ MENUDOWN (called by the mouse driver) lays down the menu box over an area of the hires screen.

☐ MENUPLT does the actual plotting of 'forms' in the menu box, taking into account, positioning, grouping (of 'image$-2' forms) and alternate command-key special imagery.

☐ MENUKEY is called by the mouse driver if the 'Commodore key' is pressed. The keycodes (if any) in the menu list are scanned starting at #MN=1 and the first match is noted by storing the current menu number and menu item number in (MNUM) and (MITM), respectively.

☐ MENUAWAY is the clean-up routine (opposite to MENUDOWN) which swaps back the area of the hires screen hidden by the menu box and un-highlights the menu name.

☐ MENUCTRL is the main routine that handles item selection with the mouse. It is basically two loops. The first locates the form to which the mouse cursor is pointing and if within the items 'area of influence' highlights it. The second checks to see if the mouse leaves the items 'area of influence' and un-highlights it, returning to the first loop. Again, the GETLIST routine simplifies the differentiation of the three graphic forms. Both loops call CHECKBOX to make sure the mouse is still within the menu box. CHECKBOX incorporates a DELAY in the speed of the mouse of 16ms. to match the normal (outside the menu box) speed. If the DELAY is reduced to say, 8ms. the mouse moves twice as fast in the menu box. Releasing the menu button over a highlighted item causes the item to FLASH. The current menu number and menu item number are stored in (MNUM) and (MITM), respectively.

2. Window Routines

Both the menu bar and menu box require windowing – that is, the area of the hires screen, hidden by the Menu System must be saved and later, restored. The Menu System uses a 'save' buffer area for the hires bitmap from $9000 to $A000 (BMAP0) that is, 1/2 of a hires screen. A corresponding color memory save area at $8E00 (CMEM0) is also needed, so that the Menu System has its own distinctive colors. The bytes are 'packed' into the buffer areas on a last-in first out basis – a buffer pointer, "BPTR", keeps track of this.

☐ OPENWNDW sets up a viewport and does the swap.

☐ CLOSWNDW is shorter and simply swaps back.

The buffer pointer routine GETBP points to the bitmap save area. GETCBP computes the equivalent pointer into the color memory save area.

3. Viewport Routines

The work horse of windowing is a series of routines which manipulate a byte-aligned viewport (rectangular region of the hires screen).

☐ VIEWPORT gets the MENU DATA (dimensions) into zero page variables, LC,TR,WD and DP for faster execution.

☐ VADDR gets the start byte of the hires bitmap (top-left corner of the rectangular region).

☐ VCPOSN positions the color memory pointers to the start of the viewport.

☐ VCOCMEM pre-colors the save area so..

☐ VSWCMEM just swaps the save area color memory with the on-screen color memory.

☐ VWIPE pre-clears the save area so ..

☐ VXFER just exchanges the hires save area with the on-screen hires bitmap.

All of the viewport routines use byte boundaries (because color memory is aligned to them, absolutely) to speed up viewport transformations.

☐ VROW handles one row (8 bytes) over the viewport width, WD.

☐ VBORDER draws a one pixel border around the current viewport (only used by menu box) as a series of bytes, so is much faster than a general line drawing algorithm.

☐ VBTOP draws the top line for a viewport (not used)

☐ VBLIN draws any horizontal line.

4. Plotting Routines

The menu routines used the general PLTFORM routines. Now it's time to get specific!

☐ PLTFORM sorts out the three graphic forms, recognized by the Menu System.

☐ PLTTEXT handles 'text$' as a series of bytes to be translated by EXCHAR into an 8 byte 'cell'.

☐ PLTIMG1 plots the one-row image directly, bypassing EXCHAR 'text$' table look-up.

☐ PLTIMG2 plots the two-row image one row at a time, by saving column positions and re-positioning as needed.

☐ PLTMARK plots the checkmark imagery.

☐ PLTALTK first finds the right edge of the viewport, moves left, then plots the Commodore-logo imagery and the uppercase alternate key.

The EXCHAR routine maintains pointers to the bitmap through the use of some 'crsr' routines, which are tempered by the viewport (i.e. knows the edges of the viewport, so can't plot outside the viewport). The PUTBYTS routine actually puts the bytes (pointed to by $A9/$AA) on the hires screen. If the DIMFlag is set then the plotting is 'dimmed' by logic ANDing the bytes with the DIM bits mask.

5. Mouse Support Routines

These routines manage the cursor and set up the interrupt driver. MOUSE requires the use of the defacto Microsoft/Amiga standard left and right buttons. The left button is called the 'Select' button and the right button is the 'Menu' button. (See 'A Two Button Mouse' Transactor Vol. 7 Iss. 06, Pg.36 for details). To activate the Menu System, the user presses the menu button (moving the mouse cursor over a menu name within the menu bar) and while still keeping it pressed, moves the mouse cursor in the list of menu items that appear in the menu box, below the menu name. Release the menu button while over a highlighted item to select, or move out of the menu box to cancel.

SETTRANS, SETARROW and SETCROSS enable the built-in default cursors. Note that since the sprite area is pre-cleared, only the most significant bytes need be submitted in 'cursor$'.

☐ MOUSESTR sets the bytes found in 'cursor$' into the sprite definition area - SLOT1. [Note 'HIRES' operates in Bank 3]. There are two more SLOTs available, but unused: SLOT2 = $FF80 (#254) and SLOT3 = $FFC0 (#255).

☐ MOUSEPTR sets the sprite pointer to SLOT1 (#253).

☐ MOUSEOFF puts back the old IRQ routine and turns off the sprite.

☐ MOUSEON sets up the new IRQ routine and turns the sprite on.

☐ MOUSECOL sets the sprite color.

☐ MSDIRQ is the new mouse driver, which moves the sprite, handles the mouse buttons and checks for optional alternate key codes (if implemented). This new IRQ routine is added to the existing old IRQ routine, so the mouse is updated every 16ms (which is fast enough to simulate smooth motion). It is also possible for the enterprising programmer to mix polled interrupts with true interrupts and CIA #1 allows the frequency of interrupts to be changed.

MSEVENT checks the mouse buttons and if pressed, branches to..

☐ SELECT if left button pressed (currently no action taken), or..

☐ MENUBTN for the right button. [Note that it's possible to make the left button the menu button by slight changes to MSEVENT and SELECT and RDMOUSE -(change line 1832 LDA #$00 . . .;logic low to LDA #$20 . . .;a logic high always)].

The mouse button status, MB, is polled every interrupt. Only if the Menu System is activated is MB consumed (set to 0), otherwise it's status reflects real time.

More of the system stack is used by the new mouse driver, 16 bytes more if the Menu System is activated. Normally, this won't bother Basic, unless you are a FOR/NEXT or GOSUB junkie.

The mouse 'position' is converted to equivalent 'cells' - columns & rows - by MCMXY, to speed up and ease the item selection process.

6. Sprite Routines

The sprite with the highest priority, sprite0, is commandeered for the job of cursor.

☐ RDMOUSE is the main routine that updates the mouse in Control Port 2, position (MX,MY) and button status (MB). [Note that location (MYI) holds 199-MY for use with non-Cartesian coordinates if need].

☐ RMPOS0Y positions the sprite0-y register below the menu bar (i.e into the menu box).

☐ RMSPRT0 initializes sprite0 hires mode, coordinates (if needed), priority and display enable.

☐ RMSCOL0 just stores the color parameter.

☐ RMBTNS tests the status of the buttons and sets the Z-flag.

There, then, is my own subjective interpretation of a Menu System. I now possess much greater understanding of and compassion for other menu systems and their designers (the Amiga comes to mind) but using this source code you might want to make changes in the design criteria! (Go ahead, make my day!)

Menus Demo Program

```
EH  100 rem save "0:menus demo",8
PC  110 rem 'hires menus demonstration
CN  120 if peek(49153)<>194 then load "hires",8,1
JN  130 if peek(32769)<> 97 then load "menus",8,1
AA  140 :
FM  150 poke52,128:poke56,128:clr:rem reserve memory
EB  160 :
PO  170 rem 'hires' variables
LA  180 hires=49152:draw=hi+3:plot=dr+3
EF  190 mve=pl+3:clscr=mv+3:dmode=cl+3
MF  200 selpc=dm+3:colour=se+3:box=co+3
BC  210 text=bo+3:prnt=te+3:chset=pr+3
LC  220 trap=ch+3
KF  230 :
NG  240 rem 'menus' variables
KJ  250 menus=32768
PI  251 mnum=me+3
BK  252 mitm=mn+3
BH  253 mflg=mn+2
GD  254 mouse=mi+3
MN  255 mb=mo+3
DH  256 mx=mb+3:my=mx+3
IH  260 :
CI  270 :
MM  280 rem user demo - experiment with it
BJ  285 :
II  290 rem c-1350 mouse in control port 2
LJ  295 :
AC  300 sys hires,0,3,0:poke53280,14
AB  310 tt$="Workbench  Version 1.0  30717 bytes
        free"
CM  320 sys prnt,0,0,chr$(14)+tt$
FN  325 sys prnt,12,10,"Working . . ..."
GJ  330 gosub1000 'build the strings
EL  335 sys prnt,12,10,"[13 spcs]"
FH  340 sys menus clr:rem clear menu lists
OE  350 sys mouse,1  :rem set arrow cursor
HN  355 :
LF  360 rem build the menus
AO  370 sys menus,1,0,1," File ",1
OI  371 sys menus,1,1,1," Open"
EN  372 sys menus,1,2,1," Show Info. . ."
EP  373 sys menus,1,3,0,"----------------"
CF  374 sys menus,1,4,1," New Folder. . ."
AM  375 sys menus,1,5,1," Close Folder"
DO  376 sys menus,1,6,1," Close Window"
EA  377 sys menus,1,7,0,"----------------"
MC  378 sys menus,1,8,1," Format. . ."
PO  379 :
HB  380 sys menus,2,0,1," Edit ",8
BM  381 sys menus,2,1,1,edit$, 192or1
DD  382 sys menus,2,2,1,type$, 192or5
GA  383 sys menus,2,3,1,pen$,  192or1
PC  384 sys menus,2,4,1,erase$,192or5
LN  385 sys menus,2,5,1,fill$, 192or1
CP  386 sys menus,2,6,1,cube$, 192or5
OH  387 sys menus,2,7,1,spray$,192or1
AN  388 sys menus,2,8,1,actn$, 192or5
JP  389 :
HJ  390 sys menus,3,0,1,faunts$,128or15
HB  391 sys menus,3,1,1," Plain",41
JH  392 sys menus,3,2,1,italic$,128or33
FL  393 sys menus,3,3,1,script$,128or13
HD  394 sys menus,3,4,0,"==================="
CH  395 sys menus,3,5,2," Align Left",42
FO  396 sys menus,3,6,1," Align Middle ",36
DJ  397 sys menus,3,7,1," Align Right",17
CA  398 :
DA  399 :
FL  400 sys menus,4,0,1," Goodies ",23
AO  401 sys menus,4,1,1,disk$, 192or1
MC  402 sys menus,4,2,1,paper$,192or1
KF  403 sys menus,4,3,2,mouse$,192or1
KO  404 sys menus,4,4,1,cmlg$, 192or1
OL  405 sys menus,4,5,0," "
EM  406 sys menus,4,6,1,"  Key  "
LA  407 :
MA  408 :
NA  409 :
GF  500 rem display menu bar & white mouse
FJ  510 sys menus on,7,6:sys mouse on,1
MH  520 :
NA  530 rem main program loop start
AJ  540 :
IE  550 rem poll variables as needed
FH  560 nm=peek(mn):im=peek(mi):bt=peek(mb)
FD  570 xm=peek(mx)+256*peek(mx+1)
LC  580 ym=peek(my)
CM  590 :
MP  600 rem 'on nm gosub xxx,xxx,xxx,xxx'
BD  610 if nm then print nm,im:poke(mn),0
AO  620 :
JG  630 rem if bt=1 then 'act on right btn
AG  640 if bt=255 goto720 'act on left btn
OP  650 :
IF  660 rem try sys draw,rnd(1)*320,rnd(1)*180
OH  670 rem try if bt=255 then sys draw,xm,ym
MB  680 :
MI  690 goto530 'main loop
AD  700 :
IH  710 remove mouse,menu bar, await keypress
NB  720 sys mouse off:sys menus off
EL  730 wait198,1:get a$:sys text:end
IF  740 :
DJ  799 :
CB  999 rem subroutines to build strings
LD  1000 ns=48:gosub2000:edit$=a$
HK  1002 ns=48:gosub2000:type$=a$
NG  1004 ns=48:gosub2000:pen$=a$
LA  1006 ns=48:gosub2000:erase$=a$
IF  1008 ns=48:gosub2000:fill$=a$
JE  1010 ns=48:gosub2000:cube$=a$
BI  1012 ns=48:gosub2000:spray$=a$
EG  1014 ns=48:gosub2000:actn$=a$
GL  1016 ns=56:gosub2000:faunts$=a$
PF  1018 ns=56:gosub2000:italic$=a$
EM  1020 ns=56:gosub2000:script$=a$
NH  1022 ns=48:gosub2000:disk$=a$
IE  1024 ns=48:gosub2000:paper$=a$
JG  1026 ns=32:gosub2000:mouse$=a$
NG  1028 ns=48:gosub2000:cmlg$=a$
CC  1030 return
AD  2000 a$="":for i=1 to ns:read byte
PH  2002 a$=a$+chr$(byte):next
```

```
CC 2004 f=fre(" "):return
KE 2006 :
OM 2999 rem string data
DH 3000 rem edit$ - [image$-2]
IN 3005 data   0, 124,  64,  64,  64,  64,   0,   0
AJ 3010 data   0,   0,   0,   0,   0,   0,   0,   0
PC 3015 data   0,  62,   2,   2,   2,   2,   0,   0
NO 3020 data   0,   0,  64,  64,  64,  64, 124,   0
PJ 3025 data   0,   0,   0,   0,   0,   0,   0,   0
KD 3030 data   0,   0,   2,   2,   2,   2,  62,   0
AA 3035 rem type$ - [image$-2]
EO 3040 data   0,   7,  15,  28,   0,   0,   0,   0
PE 3045 data   0, 255, 255,  24,  24,  24,  24,  24
NC 3050 data   0, 224, 240,  56,   0,   0,   0,   0
NL 3055 data   0,   0,   0,   0,   0,   0,   0,   0
DA 3060 data  24,  24,  24,  24,  24,  60, 126,   0
HM 3065 data   0,   0,   0,   0,   0,   0,   0,   0
OB 3070 rem pen$ - [image$-2]
BN 3075 data   0,   0,   0,   0,   0,   0,   0,   0
NO 3080 data   0, 254, 254, 130, 146, 178, 162, 162
LN 3085 data   0,   0,   0,   0,   0,   0,   0,   0
PE 3090 data   0,   0,   0,   0,   0,   0,  10,   0
JA 3095 data 130, 146, 238,  68,  40,  16, 176,   0
KO 3100 data   0,   0,   0,   0,   0,   0,   0,   0
CO 3105 rem erase$ - [image$-2]
EP 3110 data   0,   0,   0,   0,   0,   0,   0,   0
JC 3115 data   0,   0,   0,  15,  31,  33,  66, 132
GB 3120 data   0,   0,   0, 224, 160,  64,  64, 128
CC 3125 data   1,   3,   4,   4,   3,   0,   0,   0
ML 3130 data   9, 242,  12,   8, 240,   0,   0,   0
NA 3135 data   0,   0,   0,   0,   0,   0,   0,   0
JA 3140 rem fill$ - [image$-2]
IC 3145 data   0,   0,   0,   0,   0,   1,   3,   7
KO 3150 data   0,  48,  32,  64, 128,   0, 255, 255
EJ 3155 data   0,   0,   0,   0,   0,  56, 216, 152
FC 3160 data  15,  31,  15,   3,   0,   0,   0,   0
JF 3165 data 255, 254, 252, 248, 240,  32,   0,   0
KC 3170 data  24,   8,   8,   8,   8,   0,   0,   0
LD 3175 rem cube$ - [image$-2]
LN 3180 data   0,   0,   3,  12,  16,  28,  23,  26
LA 3185 data  48, 204,   3,   0,   0,   1,   7, 223
EM 3190 data   0,   0,   0, 192,  96, 224, 224, 224
GG 3195 data  21,  26,  21,  14,   3,   0,   0,   0
GC 3200 data 127, 191, 127, 191, 126, 248,  32,   0
MG 3205 data 224, 224, 224, 128,   0,   0,   0,   0
GJ 3210 rem spray$ - [image$-2]
BG 3215 data   0,   0,   1,   6,   8,  24,  16,  16
GJ 3220 data   0,   0, 255, 126,  24, 126, 153, 153
HI 3225 data   0,  16,  64,   8,  32,   8,   0,   0
HJ 3230 data  16,  16,   0,   0,   0,   0,   0,   0
JN 3235 data 153, 153, 255, 255, 255, 255,   0,   0
GH 3240 data   0,   0,   0,   0,   0,   0,   0,   0
HF 3245 rem actn$ - [image$-2]
FO 3250 data   0,   0,   0,   0,   0,   0,   0,  12
BK 3255 data   0,   0,   1,   3,   7,  14,  29,  59
LB 3260 data   0, 192, 192, 128,   0,   0, 128,  96
FM 3265 data  14,  15,   7,   3,   3,   1,   0,   0
PA 3270 data 118, 185, 221, 222, 191, 255, 255, 255
AP 3275 data 216, 180, 108, 216,  48, 160, 192, 128
NO 3280 rem faunts$ - [image$-1]
DK 3285 data   0,   0,   0,   0,   0,   0,   0,   0
AP 3290 data 126, 144,  80,  30,  80, 144, 144,  96
MK 3295 data   0,   0, 120, 206, 205, 204, 120,   0
EN 3300 data   0,   0,  56, 100, 164,  37,  66,   0
CB 3305 data   0,  48, 252,  48,  48, 241,  14,   0
LB 3310 data   0,  24,  36,  68, 130,   3,  60,   0
BM 3315 data   0,   0,   0,   0,   0,   0,   0,   0
NO 3320 rem italic$ - [image$-1]
LM 3325 data   0,   0,   0,   0,   0,   0,   0,   0
IL 3330 data  30,  12,  12,  12,  24,  24,  60,   0
OP 3335 data   0,  12,  63,  12,  24,  24,  14,   0
OH 3340 data   0,   0,  30,   3,  62, 102,  62,   0
AB 3345 data   0,  28,  12,  12,  24,  24,  60,   0
NH 3350 data   0,  12,   0,  28,  24,  24,  60,   0
HL 3355 data   0,   0,  30,  48,  96,  96,  60,   0
DF 3360 rem script$ - [image$-1]
DP 3365 data   0,   0,   0,   0,   0,   0,   0,   0
PL 3370 data 114, 140, 192,  56,   6, 113, 129, 126
ID 3375 data   0,   0, 112, 200, 192, 193, 126,   0
HB 3380 data   0,   0,  80, 104, 200,  69,  69,   0
HB 3385 data  24,   0,  24,  24,  24, 249,  30,   0
CB 3390 data   0, 176, 200, 136, 200, 177, 142, 128
MG 3395 data   0,  48, 252,  48,  48, 241,  14,   0
EB 3400 rem disk$ - [image$-2]
HI 3405 data   0, 127,  64,  64,  64,  64,  64,  64
CF 3410 data   0, 255,   0,   0, 126, 129, 129, 129
BF 3415 data   0, 254,   2,   2,   2,   6,   2,   2
OK 3420 data  64,  64,  64,  64,  64,  64, 127,   0
DC 3425 data 126,   0,  24,  24,  24,   0, 255,   0
CG 3430 data   2,   2,   2,   2,   2,   2, 254,   0
FK 3435 rem paper$ - [image$-2]
FH 3440 data   0,  15,  16,  32,  32,  32,  16,   8
OE 3445 data   0, 255,   0,   0,   0,   0,   0,   0
DG 3450 data   0, 248,  16,  32,  32,  32,  16,   8
AK 3455 data   4,  30,  33,  65,  65,  34,  31,   0
JG 3460 data   0,   0,   0,   0,   0,   0, 255,   0
LJ 3465 data   8,   4,   2,   2,   2,   4, 248,   0
BP 3470 rem mouse$ - [image$-2]
AN 3475 data   0, 127,  81,  81,  81,  95,  72,  72
CO 3480 data   0, 255,  69,  69,  69, 125,   9,   9
OO 3485 data  72,  72,  72,  79,  80,  96, 127,   0
FN 3490 data   9,   9,   9, 249,   5,   3, 255,   0
BH 3495 rem cmlg$ - [image$-2]
HA 3500 data   0, 127,  64,  65,  71,  79,  79,  95
KM 3505 data   0, 255,   0, 252, 252, 255, 131,   3
DM 3510 data   0, 254,   2,   2,   2, 226, 194,   2
LD 3515 data  95,  79,  79,  71,  65,  64, 127,   0
HK 3520 data   3, 131, 255, 252, 252,   0, 255,   0
IM 3525 data   2, 194, 226,   2,   2,   2, 254,   0
CF 3530 end of data
```

Program to generate pointer sprite for optional editing

```
MA 100 rem pointer arrow sprite generator
BF 110 for j=1 to 64 : read x
NA 120 ch=ch+x : next
BC 130 if ch<>499 then print "checksum error" : end
BE 140 print "data ok, now creating file" : print
OK 150 restore
LG 160 open8,8,8,"0:arrow.spr,p,w"
MD 170 print#8,chr$(192)chr$(63);
HJ 180 for j=1 to 64 : read x
OD 190 print#8,chr$(x); : next
AL 200 close 8
NK 210 print "prg file 'arrow.spr' created. . .
GN 220 print "this generator no longer needed.
```

```
EA 230 rem
LD 240 data  0,  0,  0,  0,  0,  0, 32,  0
EM 250 data  0, 48,  0,  0, 56,  0,  0, 60
FA 260 data  0,  0, 62,  0,  0, 63,  0,  0
GB 270 data 60,  0,  0, 38,  0,  0,  6,  0
CP 280 data  0,  3,  0,  0,  3,  0,  0,  0
AP 290 data  0,  0,  0,  0,  0,  0,  0,  0
KP 300 data  0,  0,  0,  0,  0,  0,  0,  0
KH 310 data  0,  0,  0,  0,  0,  0,  0, 68
```

Generator for "MENUS"

```
CC 100 rem save "0:menus.bas",8
AN 110 rem a hires menu system for use
KP 120 rem with 'hires' & 'viewports'
EB 130 rem   by anthony bryant
BA 140 rem   winnipeg,manitoba
KA 150 :
GD 160 for j=1to3030:read x:ch=ch+x:next
KM 170 if ch<>347286 then print "checksum error!":stop
MM 180 restore
LG 190 rem data ok, create ml file
GI 200 open 15,8,15
NO 210 open 8,8,1,"0:menus"
OM 220 input#15,e,e$,b,c
LI 230 if e then close15:print e;e$;b;c:stop
AH 240 print#8,chr$(0)chr$(128);
KP 250 for j=1to3030:read x:print#8,chr$(x):next
FH 260 close8:close15
PM 270 print "** module created **":end
MI 280 :
GP 1000 data  76,  97, 128,   0,   1,   0,   0,   0,   0
NN 1010 data  76, 152, 137,   0,   0,   0,   0,   0,   0
CL 1020 data   0,   0,   0,   0,   0,   0,   0,   0,   0
ML 1030 data   0,   0,   0,   0,   0,   0,   0,   0,   0
FJ 1040 data   1,   1,   1,   1,   1,   1,   1,  40,   0
DN 1050 data   0,   0,   0,   0,   0,   0,   1,   0,   0
KN 1060 data   0,   0,   0,   0,   0,   0,   0,   0,   0
EO 1070 data   0,   0,   0,   0,   0,   0,   0,   0,   0
OO 1080 data   0,   0,   0,   0,   0,   0,   0,   0,   0
GC 1090 data   0,   0, 128,  64,  32,  16,   8,   4,   2
ON 1100 data   1,   0, 216,   0,   0,   0,   0,  32, 121
IA 1110 data   0, 240, 111, 201,  44, 240,  16,  72,  32
BI 1120 data   3, 129, 104, 201, 156, 240,  77, 201, 145
PJ 1130 data 240, 112,  76,   8, 175,  32, 241, 183, 138
FK 1140 data 240,  61, 224,   8, 176,  57, 142,   3, 128
IM 1150 data  32, 241, 183, 224,   9, 176,  47, 142,   6
DL 1160 data 128,  32, 241, 183, 224,   3, 176,  37, 142
JP 1170 data   7, 128,  32,  33, 129,  32, 121,   0, 208
JO 1180 data   1,  96,  32, 253, 174,  32, 158, 173,  32
JM 1190 data 163, 182,  32, 172, 129,  32, 121,   0, 208
PM 1200 data   1,  96,  32, 241, 183, 138,  76, 224, 129
PN 1210 data  76,  72, 178, 162,   0, 138, 157,  19, 128
FM 1220 data 232, 224,   8, 144, 248, 141,   8, 128, 170
CJ 1230 data 157,   0, 141, 232, 208, 250, 173,   5, 128
JP 1240 data 240,  37,  32,  56, 130, 140,   6, 128,  32
FM 1250 data 187, 133, 169,   0, 240,  24, 173,   5, 128
HE 1260 data 208,  19,  32, 255, 133, 173,  19, 128, 240
FP 1270 data  11,  32,  19, 130,  32, 166, 133,  32,  62
DB 1280 data 130, 169, 255, 141,   5, 128,  96,  32, 115
KG 1290 data   0, 208,   1,  96,  32, 241, 183, 138,  41
GP 1300 data  15, 141,   4, 128,  32, 241, 183, 138,  10
NN 1310 data  10,  10,  10,  13,   4, 128, 141,   4, 128
FA 1320 data  96, 172,   3, 128, 174,   6, 128, 208,   5
HH 1330 data 160,   0, 174,   3, 128, 189,  82, 128,  73
PI 1340 data 255,  57,  59, 128, 153,  59, 128, 189,  82
FI 1350 data 128,  73, 255,  57,  67, 128, 153,  67, 128
DD 1360 data 173,   7, 128, 201,   0, 240,   5, 201,   2
HG 1370 data 240,  11,  96, 185,  59, 128,  29,  82, 128
OJ 1380 data 153,  59, 128,  96, 185,  67, 128,  29,  82
BM 1390 data 128, 153,  67, 128,  96, 172,   3, 128, 174
AH 1400 data   6, 128, 208,   5, 160,   0, 174,   3, 128
GN 1410 data 185,  59, 128,  61,  82, 128, 141,  95, 128
IA 1420 data 185,  67, 128,  61,  82, 128,  96, 174,   6
FP 1430 data 128, 157,  74, 128,  96, 174,   6, 128, 188
HG 1440 data  74, 128,  96, 173,   3, 128,  56, 233,   1
LK 1450 data  10,  10, 133, 163,  10, 101, 163, 133, 163
DN 1460 data  10, 101, 163, 133, 163, 173,   6, 128,  10
JD 1470 data  10, 101, 163, 168,  96, 170,  32, 143, 129
LE 1480 data 138, 153,   0, 141, 200, 165,  34, 153,   0
CC 1490 data 141, 200, 165,  35, 153,   0, 141, 200, 140
LJ 1500 data   8, 128, 169,   0,  32, 224, 129, 174,   3
KB 1510 data 128, 236,  19, 128, 144,   3, 142,  19, 128
CN 1520 data 173,   6, 128, 221,  19, 128, 144,   3, 157
JC 1530 data  19, 128,  96, 172,   8, 128, 153,   0, 141
EB 1540 data  96,  32, 143, 129, 190,   0, 141, 200, 185
OL 1550 data   0, 141, 133, 163, 200, 185,   0, 141, 133
MO 1560 data 164, 200, 185,   0, 141, 133, 165, 140,   8
PG 1570 data 128, 138,  36, 165,  16,   6,  80,   1,  74
ED 1580 data  74,  74,  74, 170, 165, 165,  36, 165,  96
MH 1590 data 169,   1, 141,  51, 128, 160,   0, 140,   6
LH 1600 data 128, 200, 140,   3, 128, 204,  19, 128, 240
ND 1610 data   2, 176,  16,  32, 231, 129, 112,   6, 172
EA 1620 data   3, 128, 200, 208, 235, 169,   2, 141,  51
MI 1630 data 128, 160,   0, 140,   3, 128,  96, 174,  27
CM 1640 data 128, 134, 189, 172,  35, 128, 132, 190,  32
AE 1650 data 213, 136, 160,   0, 140,   6, 128, 200, 140
EE 1660 data   3, 128, 204,  19, 128, 240,   2, 176, 221
JA 1670 data 185,  19, 128, 208,   6, 141,   7, 128,  32
BJ 1680 data  33, 129,  32, 157, 136,  32, 100, 129,  32
FI 1690 data 231, 129,  41,  63, 240,   9,  24, 101, 189
JN 1700 data 170, 164, 190,  32, 213, 136,  32, 231, 129
GH 1710 data   5, 158,  32, 224, 129,  36, 165,  32, 200
BK 1720 data 135, 172,   3, 128, 200, 208, 194, 173,   5
DG 1730 data 128, 240,  99, 173,  19, 128, 240,  94, 172
FN 1740 data  35, 128, 132, 190,  32, 112, 138, 200, 173
BF 1750 data  51, 128, 136, 196, 190, 240,   5,  74, 144
ON 1760 data 248, 176,  72, 160,   0, 140,   6, 128, 200
DL 1770 data 140,   3, 128, 204,  19, 128, 240,   2, 176
MO 1780 data  56,  32, 231, 129, 133, 192,  41,  63, 133
IC 1790 data 189, 205,  13, 128, 240,   2, 176,  40, 138
BD 1800 data  24, 101, 189, 205,  13, 128, 240,   2, 176
MP 1810 data   6, 172,   3, 128, 200, 208, 213, 134, 180
LK 1820 data  32, 100, 129, 173,  95, 128, 208,  13, 165
PJ 1830 data 180, 166, 189, 164, 190,  36, 192,  32, 118
MH 1840 data 132,  24,  96,  32,  56, 130,  56,  96, 165
JH 1850 data 190,  24, 109,  51, 128, 133, 177, 165, 189
MD 1860 data 133, 176, 160,   0, 132, 178, 132, 179, 132
DD 1870 data 166, 162,   1, 142,   6, 128,  32, 231, 129
LA 1880 data 112,  12,  41,  63, 240,   2, 232, 232, 160
GF 1890 data   0, 132, 166, 240,  27,  41,  63, 197, 166
LF 1900 data 240,   2, 176,   4, 160,   0, 132, 166, 164
MJ 1910 data 166, 133, 166, 138,  56, 101, 166, 170, 192
FK 1920 data   0, 208,   4, 230, 179, 230, 179, 228, 178
KJ 1930 data 144,   2, 134, 178, 238,   6, 128, 172,   3
JP 1940 data 128, 173,   6, 128, 217,  19, 128, 144, 190
```

```
GA 1950 data 240, 188, 165, 176,  24, 101, 178, 201,  39
KJ 1960 data 144,  11, 240,   9, 233,  39,  73, 255, 101
CN 1970 data 189,  76,   6, 131, 165, 176, 153,  27, 128
EF 1980 data 165, 178, 153,  43, 128, 165, 177, 153,  35
PN 1990 data 128, 165, 179, 201,  21, 144,   2, 169,  21
BP 2000 data 153,  51, 128,  96,  32, 252, 130,  32, 166
EO 2010 data 133,  32,  96, 135, 164, 177, 132, 159, 162
LB 2020 data   1, 142,   6, 128, 160,   0, 132, 166,  32
JO 2030 data 157, 136,  32, 231, 129,  80,  31,  41,  63
OA 2040 data 197, 166, 240,   2, 176,   4, 160,   0, 132
KB 2050 data 166, 164, 166, 133, 166,  24, 101, 176, 170
EA 2060 data 192,   0, 240,   9, 164, 159, 136, 136,  76
HB 2070 data 201, 131, 166, 176, 164, 159,  32, 213, 136
KE 2080 data 165, 159,  32, 129, 129,  32, 100, 129, 240
JC 2090 data   3,  32,  40, 136,  32, 231, 129,  32, 200
DE 2100 data 135,  36, 165, 112,  20, 160,   0, 132, 166
AK 2110 data 165, 165,  41,  63, 240,  13, 170, 189, 194
ID 2120 data 235,  32,  56, 136,  76, 250, 131,  32, 208
DG 2130 data 136,  32, 208, 136, 238,   6, 128, 172,   3
KM 2140 data 128, 173,   6, 128, 217,  19, 128, 144, 147
JM 2150 data 240, 145,  96, 165, 197, 201,  64, 208,   1
NL 2160 data  96, 160,   1, 140,   3, 128, 204,  19, 128
FF 2170 data 240,   2, 176,  67, 185,  19, 128, 240,  45
NF 2180 data 162,   0, 142,   6, 128,  32, 100, 129, 173
JK 2190 data  95, 128, 208,  32,  32, 231, 129, 112,  11
BM 2200 data 174,   6, 128, 240,   6,  41,  63, 197, 197
FL 2210 data 240,  22, 238,   6, 128, 172,   3, 128, 173
LP 2220 data   6, 128, 217,  19, 128, 144, 226, 240, 224
HB 2230 data 172,   3, 128, 200, 208, 190, 198, 198,  32
PO 2240 data 100, 129, 173,  95, 128, 208,   1,  96,  32
GP 2250 data  56, 130, 140,   6, 128,  96,  32, 187, 133
CG 2260 data 165, 180, 166, 189, 164, 190,  36, 192,  16
IA 2270 data   4,  80,   2, 112,  31, 133, 191,  32, 110
BM 2280 data 134, 160,   0, 177, 247,  10,  10,  10,  10
GF 2290 data 133, 175, 177, 247,  74,  74,  74,  74,   5
FC 2300 data 175, 145, 247, 200, 196, 191, 144, 233,  96
BC 2310 data 133, 191,  32, 110, 134, 162,   0,  32, 129
CJ 2320 data 132,  32, 143, 134, 232, 224,   2, 144, 245
OJ 2330 data  96,  32,  77, 139, 164, 177, 132, 159, 169
GE 2340 data   1, 141,   6, 128, 166, 178, 134, 191, 166
DK 2350 data 176, 134, 158,  32, 136, 129, 132, 159,  32
IF 2360 data 231, 129,  80,   9, 134, 191,  41,  63,  24
MJ 2370 data 101, 176, 133, 158,  32, 129, 133, 176,  45
BI 2380 data  32, 138, 139, 240, 102,  36, 165,  80,  26
NO 2390 data 196, 159, 240,   9, 144,  76, 136, 196, 159
CH 2400 data 240,   2, 176,  69, 228, 158, 144,  65, 138
DF 2410 data 229, 158, 197, 191, 176,  58, 144,   4, 196
MK 2420 data 159, 208,  52,  32, 100, 133,  32, 129, 133
CJ 2430 data 176,  60,  32, 138, 139, 240,  64,  36, 165
CI 2440 data  80,  26, 196, 159, 240,   9, 144,  24, 136
IB 2450 data 196, 159, 240,   2, 176,  17, 228, 158, 144
IL 2460 data  13, 138, 229, 158, 197, 191, 176,   6, 144
LA 2470 data 216, 196, 159, 240, 212,   8,  32, 100, 133
CI 2480 data  40, 176,   8, 206,   6, 128, 173,   6, 128
KM 2490 data 208,   3, 238,   6, 128,  76, 186, 132, 162
FD 2500 data   0, 142,   6, 128, 142,  12, 128,  96,  32
HK 2510 data 100, 129, 173,  95, 128, 208, 239,  32,  89
CN 2520 data 133,  32,  92, 133,  32,  95, 133, 160,  48
OC 2530 data  32, 120, 133,  32, 100, 129, 173,  95, 128
PD 2540 data 240,   1,  96, 165, 191, 166, 158, 164, 159
HA 2550 data  36, 165,  76, 118, 132, 162, 184, 202, 208
II 2560 data 253, 136, 208, 248,  96, 160,  16,  32, 120
ED 2570 data 133,  32, 137, 138,  32, 112, 138, 228, 176
FN 2580 data 144,  20, 138, 229, 176, 197, 178, 176,  13
JH 2590 data 196, 177, 144,   9, 152, 229, 177, 197, 179
AH 2600 data 176,   2,  24,  96,  56,  96,  32,  42, 134
OI 2610 data  32, 243, 134,  32, 155, 134,  32, 216, 133
IM 2620 data  32, 159, 134,  32, 246, 134,  76, 249, 133
KC 2630 data  32,  42, 134,  32, 234, 133,  32, 246, 134
LE 2640 data  76, 159, 134,   0,   0,   0,   0,   0,   0
GI 2650 data 173, 199, 133, 174, 200, 133, 133, 167, 134
CH 2660 data 168,  96,  32, 205, 133, 172,   3, 128, 240
FE 2670 data   2, 160,   1, 153, 201, 133, 138, 153, 203
MA 2680 data 133,  96, 172,   3, 128, 240,   2, 160,   1
IF 2690 data 185, 201, 133, 190, 203, 133, 208,  10, 165
OL 2700 data 167, 166, 168, 208,   4, 169,   0, 162, 144
BJ 2710 data 141, 199, 133, 142, 200, 133,  96, 173, 200
HO 2720 data 133,  56, 233, 144, 133, 250, 173, 199, 133
PO 2730 data  70, 250, 106,  70, 250, 106,  70, 250, 106
KL 2740 data 133, 249,  24, 165, 250, 105, 142, 133, 250
GO 2750 data  96,   0,   0, 172,   3, 128, 185,  27, 128
PF 2760 data 133, 176, 185,  35, 128, 133, 177, 185,  43
JE 2770 data 128, 133, 178, 185,  51, 128, 133, 179,  24
OP 2780 data 165, 176, 101, 178, 133, 156,  32, 239, 136
JG 2790 data 166, 249, 142,  40, 134, 141,  41, 134,  96
FA 2800 data 173,  40, 134, 174,  41, 134, 133, 167, 134
IK 2810 data 168,  72, 169, 127, 141,  13, 220, 169,  52
BF 2820 data 133,   1, 104,  96, 166, 176, 164, 177, 169
PJ 2830 data   0, 133, 247, 133, 248, 152, 240,   6,  32
LP 2840 data 143, 134, 136, 208, 250, 138,  24, 101, 247
NH 2850 data 133, 247, 170, 169,   0, 101, 248,  72, 105
ND 2860 data 204, 133, 248, 104,  96,  24, 165, 247, 105
EI 2870 data  40, 133, 247, 144,   2, 230, 248,  96, 173
HL 2880 data   4, 128,  44, 169,   0, 133, 163,  32,  94
LP 2890 data 134,  32, 106, 134,  32,  10, 134, 162,   0
JD 2900 data 160,   0, 165, 163, 208,   8, 177, 247,  72
PI 2910 data 177, 249, 145, 247, 104, 145, 249, 200, 196
JJ 2920 data 178, 144, 237, 152,  24, 101, 249, 133, 249
OI 2930 data 144,   2, 230, 250,  32, 143, 134, 232, 228
JA 2940 data 179, 144, 217,  76,  19, 135,  24, 165, 171
ND 2950 data 105,  64, 133, 167, 165, 172, 105,   1, 133
EO 2960 data 168,  24, 165, 173, 105,  64, 133, 169, 165
GI 2970 data 174, 105,   1, 133, 170,  96, 169,   0,  44
CL 2980 data 169,   1, 133, 164,  32,  84, 134, 133, 169
OK 2990 data 134, 170,  32, 205, 133, 160,   0,  32,  29
LB 3000 data 135,  32, 229, 134, 164, 175, 200, 196, 179
CF 3010 data 144, 243, 169,  55, 133,   1, 169, 129, 141
HN 3020 data  13, 220,  96, 132, 175, 162,   0, 165, 167
KB 3030 data 133, 171, 165, 168, 133, 172, 165, 169, 133
MA 3040 data 173, 165, 170, 133, 174, 160,   0, 165, 164
MG 3050 data 240,   8, 177, 169,  72, 177, 167, 145, 169
CA 3060 data 104, 145, 167, 200, 192,   8, 208, 237, 152
OK 3070 data  24, 101, 167, 133, 167, 144,   2, 230, 168
LM 3080 data 152,  24, 101, 169, 133, 169, 144,   2, 230
EL 3090 data 170, 232, 228, 178, 144, 210,  96,  32,  84
FC 3100 data 134, 162,   0, 169, 128, 160,   7, 145, 167
GK 3110 data 136,  16, 251, 232, 228, 179, 176,  14, 165
FB 3120 data 167, 105,  64, 133, 167, 165, 168, 105,   1
LP 3130 data 133, 168, 208, 228, 162,   0, 160,   7,  32
GG 3140 data 178, 135, 162,   0, 136, 169,   1, 145, 167
MD 3150 data 136,  16, 251, 232, 228, 179, 176,  17,  56
HL 3160 data 165, 167, 233,  64, 133, 167, 165, 168, 233
FH 3170 data   1, 133, 168, 160,   7, 208, 227,  76,  19
AH 3180 data 135,  32,  84, 134, 162,   0, 160,   0, 169
CJ 3190 data 255, 145, 167, 232, 228, 178, 176,  12, 165
GI 3200 data 167, 105,   8, 133, 167, 144,   2, 230, 168
```

```
AA 3210 data 208, 235,  96,  16,   4,  80,  20, 112,  53
FE 3220 data 232, 160,   0, 140,  96, 128, 202, 240,   8
IB 3230 data 177, 163,  32,  76, 136, 200, 208, 245,  96
KO 3240 data 232, 160, 128, 140,  96, 128, 165, 163, 164
DC 3250 data 164, 133, 169, 132, 170, 202, 208,   1,  96
DK 3260 data  32,  76, 136,  24, 165, 169, 105,   8, 133
DB 3270 data 169, 144,   2, 230, 170,  76, 238, 135, 165
LO 3280 data 158, 133, 155, 138,  72, 232,  32, 225, 135
JC 3290 data 166, 158, 164, 159, 165, 155, 134, 155, 170
BA 3300 data 200,  32, 213, 136, 104, 170, 232,  32, 238
EB 3310 data 135, 166, 155, 164, 159, 136,  76, 213, 136
EE 3320 data  36, 165, 112,   6,  32,  31, 137,  76, 131
IC 3330 data 137,  32, 131, 137,  76,  31, 137, 141,  93
LD 3340 data 128,  32, 246, 136,  32,  37, 137,  32, 157
PA 3350 data 136, 160,   0, 140,  96, 128, 173,  93, 128
JD 3360 data 141,  93, 128, 138,  72, 152,  72,  32,  91
MH 3370 data 136, 104, 168, 104, 170,  96, 173,  96, 128
CD 3380 data 208,  54, 173,  93, 128,  48,  68, 201,  32
GE 3390 data 144,  48, 201,  96, 144,   4,  41, 223, 208
PG 3400 data   2,  41,  63, 174,  94, 128, 240,   2,   9
FP 3410 data 128, 162,   0, 134, 170,  10,  38, 170,  10
NE 3420 data  38, 170,  10,  38, 170,  24, 109,  91, 128
IM 3430 data 170, 165, 170, 109,  92, 128, 168, 134, 169
MF 3440 data 132, 170,  76,  45, 137, 201,  14, 208,   7
LD 3450 data 173,  92, 128,   9,   8, 208,  30, 201,  18
FF 3460 data 240,  36,  96,  41, 127, 201, 127, 208,   2
NB 3470 data 169,  94, 201,  32, 144,   4,   9,  64, 208
BK 3480 data 186, 201,  14, 208,   9, 173,  92, 128,  41
DK 3490 data 240, 141,  92, 128,  96, 201,  18, 208,   5
AO 3500 data 169,   0, 141,  94, 128,  96, 166, 176, 164
OI 3510 data 159, 200, 134, 158, 132, 159,  32, 110, 134
PM 3520 data 133, 250, 134, 249,   6, 249,  42,   6, 249
MM 3530 data  42,   6, 249,  42,  24, 105, 224, 133, 250
AH 3540 data  96, 166, 176, 164, 177,  76, 213, 136, 166
PH 3550 data 156, 202, 202, 164, 159,  76, 213, 136,   0
KN 3560 data   1,   2,   4,  40,  16,   0,   0,  56,  70
IC 3570 data 180, 184, 180,  70,  56,   0, 170,  85, 170
NH 3580 data  85, 170,  85, 170,  85,   0,   0,   0,   0
GP 3590 data   0,   0,   0,   0, 162, 255, 160, 136, 208
EC 3600 data   4, 162,   7, 160, 137, 134, 169, 132, 170
FG 3610 data 173,  14, 220,  41, 254, 141,  14, 220, 169
OK 3620 data  49, 133,   1, 160,   7, 177, 169, 153,  23
CN 3630 data 137, 136,  16, 248, 173,  95, 128, 240,  14
BE 3640 data 160,   7, 185,  23, 137,  57,  15, 137, 153
LH 3650 data  23, 137, 136,  16, 244, 160,   7, 177, 249
AM 3660 data  25,  23, 137, 145, 249, 136,  16, 246, 169
HG 3670 data  55, 133,   1, 173,  14, 220,   9,   1, 141
CP 3680 data  14, 220, 166, 158, 232, 228, 156, 176,  13
NC 3690 data 134, 158,  24, 165, 249, 105,   8, 133, 249
NK 3700 data 144,   2, 230, 250,  96, 166, 158, 202, 228
NM 3710 data 176, 144,  13, 134, 158,  56, 165, 249, 233
KN 3720 data   8, 133, 249, 176,   2, 198, 250,  96,  32
HP 3730 data 121,   0, 240, 100, 201,  44, 240,  12,  72
EI 3740 data  32,  27, 138, 104, 201, 145, 240,  95,  76
DJ 3750 data   8, 175,  32, 253, 174,  32, 158, 173,  36
EM 3760 data  13,  48,  38,  32, 161, 183, 224,   0, 240
AN 3770 data  35, 224,   1, 240,   7, 224,   2, 240,  11
KE 3780 data  76,  72, 178, 169, 160, 160, 139, 162,  39
AD 3790 data 208,   6, 169, 199, 160, 139, 162,  15, 133
LN 3800 data  34, 132,  35, 208,   4,  32, 163, 182, 170
DA 3810 data 160,  63, 169,   0, 153,  64, 255, 136,  16
JK 3820 data 250, 160,   0, 232, 202, 240,  10, 177,  34
ID 3830 data 153,  64, 255, 200, 192,  64, 144, 243, 169
PI 3840 data 253, 141, 248, 207,  96, 162, 254, 169,  49
FO 3850 data 160, 234, 208,   6, 162,   1, 169,  42, 160
BF 3860 data 138, 120, 141,  20,   3, 140,  21,   3,  32
JG 3870 data  88, 139,  88,  96,  32, 115,   0, 208,   1
OB 3880 data  96,  32, 241, 183, 138,  41,  15,  76, 134
IK 3890 data 139, 169, 138,  72, 169,  55,  72,   8,  72
IJ 3900 data  72,  72,  76,  49, 234,  32, 137, 138,  32
AG 3910 data 138, 139, 141,  12, 128, 240,   6,  32,  89
BH 3920 data 138,  76, 129, 234, 172, 142,   2, 192,   2
CF 3930 data 208,   7, 166, 198, 240,   3,  32,  14, 132
LO 3940 data  76, 129, 234,  48,   4, 234,  16,   4,  96
BP 3950 data  96, 234, 234,  32, 143, 130, 176, 247,  32
NI 3960 data 136, 131,  32, 174, 132,  76, 107, 132, 173
PI 3970 data  16, 128,  74, 173,  15, 128, 106,  74,  74
LN 3980 data 170, 142,  13, 128, 173,  17, 128,  74,  74
KC 3990 data  74, 168, 140,  14, 128,  96, 169, 192, 141
GH 4000 data   2, 220, 169, 128, 141,   0, 220, 162,   0
AD 4010 data 232, 208, 253, 174,  25, 212, 173,   0, 220
DD 4020 data  72,  41,  16, 141,  12, 128, 138,  48,   3
AE 4030 data 169,  32,  44, 169,   0,  13,  12, 128,  74
AJ 4040 data  74,  73, 255, 141,  12, 128, 104,  41,  15
JK 4050 data 162, 255, 142,   2, 220, 162, 127, 142,   0
GO 4060 data 220, 168,  41,   1, 208,  12, 173,   1, 208
DK 4070 data 201,  48, 240,  20, 206,   1, 208, 208,  15
PN 4080 data 152,  41,   2, 208,  10, 173,   1, 208, 201
AH 4090 data 247, 240,   3, 238,   1, 208, 152,  41,   4
NM 4100 data 208,  32, 174,   0, 208, 173,  16, 208,  41
LO 4110 data   1, 208,   4, 224,  22, 240,  50, 206,   0
DL 4120 data 208, 224,   0, 208,  43, 173,  16, 208,  41
IE 4130 data 254, 141,  16, 208,  76,  40, 139, 152,  41
IF 4140 data   8, 208,  27, 174,   0, 208, 173,  16, 208
PJ 4150 data  41,   1, 240,   4, 224,  85, 240,  13, 238
FD 4160 data   0, 208, 208,   8, 173,  16, 208,   9,   1
PE 4170 data 141,  16, 208, 173,   0, 208,  56, 233,  22
NE 4180 data 141,  15, 128, 173,  16, 208,  41,   1, 233
NE 4190 data   0, 141,  16, 128, 173,   1, 208,  56, 233
EB 4200 data  48, 141,  17, 128, 169, 199, 237,  17, 128
PF 4210 data 141,  18, 128,  96, 165, 177,  10,  10,  10
OJ 4220 data 105,  52, 141,   1, 208,  96, 173,   1, 208
JI 4230 data 201,  48, 176,  24, 169,  75, 141,   0, 208
NH 4240 data 141,   1, 208, 169, 254,  45,  16, 208, 141
OO 4250 data  16, 208, 169, 254,  45,  27, 208, 141,  27
AF 4260 data 208, 138,  48,   5,  13,  21, 208, 208,   3
MJ 4270 data  45,  21, 208, 141,  21, 208,  96, 141,  39
MK 4280 data 208,  96, 169,   4,  44,  12, 128, 208,   9
AA 4290 data  10,  44,  12, 128, 208,   6, 169,   0,  96
ID 4300 data 169, 255,  96, 169,   1,  96,   0,   0,   0
PJ 4310 data   0,   0,   0,  32,   0,   0,  48,   0,   0
HD 4320 data  56,   0,   0,  60,   0,   0,  62,   0,   0
BE 4330 data  63,   0,   0,  60,   0,   0,  38,   0,   0
IM 4340 data   6,   0,   0,   3,   0,   0,   3,   0,   0
BO 4350 data 136,   0,   0,  80,   0,   0,  32,   0,   0
FB 4360 data  80,   0,   0, 136,   0,   0
```

Generator for Garry Kiziak's "HIRES"

This program, as mentioned in the article, is not necessary for the MENUS program to work. It has been included here because it's used in the demo program. You'll notice that Anthony only uses Garry's HIRES to turn on the hi-res screen, print some hi-res messages, and turn the lo-res screen back on in line 730. The messages aren't required for correct operation, and the other calls could be replaced by POKE statements. HIRES is used again in 660

and 670 of the demo, but this is merely a sample menu selection and currently disabled in REM statements. However, HIRES makes a terrific companion for this system.

```
NF 1000 rem 'hires' generator by garry kiziak
DC 1010 for j=1 to 2080 : read x
BJ 1020 ch=ch+x : next
AO 1030 if ch<>245727 then print "checksum error" : end
FM 1040 print "data ok, now creating file" : print
CD 1050 restore
IM 1060 open8,8,8,"0:hires,p,w"
EM 1070 print#8,chr$(0)chr$(192);
JG 1080 for j=1 to 2080 : read x
CM 1090 print#8,chr$(x); : next
ED 1100 close 8
KA 1110 print "prg file 'hires' created. . .
KF 1120 print "this generator no longer needed.
II 1130 rem
JB 1140 data 76, 194, 193, 76, 247, 195, 76, 98, 195
EN 1150 data 76, 110, 194, 76, 30, 194, 76, 214, 196
BB 1160 data 76, 228, 196, 76, 11, 197, 76, 67, 197
LH 1170 data 76, 169, 192, 76, 206, 197, 76, 199, 199
PP 1180 data 76, 4, 200, 0, 0, 0, 0, 0, 0
GH 1190 data 0, 0, 0, 0, 0, 0, 255, 128, 0
FI 1200 data 7, 248, 0, 0, 0, 0, 0, 0, 0
CG 1210 data 0, 0, 1, 0, 15, 240, 240, 0, 0
KL 1220 data 208, 0, 0, 0, 0, 173, 58, 192, 208
HN 1230 data 27, 173, 0, 221, 141, 57, 192, 173, 24
PH 1240 data 208, 141, 58, 192, 173, 17, 208, 141, 59
AG 1250 data 192, 173, 22, 208, 141, 60, 192, 32, 110
ML 1260 data 192, 96, 173, 0, 3, 201, 231, 208, 7
JI 1270 data 173, 1, 3, 201, 192, 240, 44, 173, 0
FM 1280 data 3, 141, 234, 192, 173, 1, 3, 141, 235
AE 1290 data 192, 169, 231, 141, 0, 3, 169, 192, 141
DA 1300 data 1, 3, 173, 2, 3, 141, 41, 193, 173
HC 1310 data 3, 3, 141, 42, 193, 169, 8, 141, 2
FC 1320 data 3, 169, 193, 141, 3, 3, 96, 173, 58
JL 1330 data 192, 240, 26, 141, 24, 208, 173, 57, 192
OP 1340 data 141, 0, 221, 173, 59, 192, 141, 17, 208
LG 1350 data 173, 60, 192, 141, 22, 208, 169, 0, 141
IH 1360 data 58, 192, 96, 72, 169, 127, 141, 13, 220
ME 1370 data 165, 1, 141, 56, 192, 41, 253, 133, 1
HF 1380 data 104, 96, 72, 173, 56, 192, 133, 1, 169
IJ 1390 data 129, 141, 13, 220, 104, 96, 16, 3, 76
HM 1400 data 139, 227, 142, 13, 3, 44, 76, 192, 16
DH 1410 data 245, 169, 0, 133, 20, 169, 0, 133, 21
IA 1420 data 162, 250, 154, 169, 167, 72, 169, 233, 72
DO 1430 data 76, 163, 168, 32, 169, 192, 173, 234, 192
JK 1440 data 141, 0, 3, 173, 235, 192, 141, 1, 3
CP 1450 data 173, 41, 193, 141, 2, 3, 173, 42, 193
AL 1460 data 141, 3, 3, 169, 0, 141, 76, 192, 76
PD 1470 data 131, 164, 164, 254, 240, 13, 160, 0, 145
BO 1480 data 251, 200, 208, 251, 230, 252, 198, 254, 208
NF 1490 data 243, 164, 253, 240, 10, 136, 240, 5, 145
OP 1500 data 251, 136, 208, 251, 145, 251, 96, 32, 201
GK 1510 data 192, 160, 0, 132, 251, 160, 204, 132, 252
LE 1520 data 160, 232, 132, 253, 160, 3, 132, 254, 32
II 1530 data 43, 193, 169, 0, 133, 251, 169, 224, 133
LF 1540 data 252, 169, 64, 133, 253, 169, 31, 133, 254
BH 1550 data 169, 0, 32, 43, 193, 76, 218, 192, 32
FG 1560 data 253, 174, 32, 138, 173, 32, 247, 183, 166
KB 1570 data 21, 165, 20, 96, 32, 253, 174, 32, 124
HL 1580 data 193, 141, 43, 192, 142, 44, 192, 32, 121
AP 1590 data 193, 141, 45, 192, 142, 46, 192, 169, 63
MN 1600 data 162, 1, 44, 53, 192, 16, 4, 169, 159
GE 1610 data 162, 0, 205, 43, 192, 138, 237, 44, 192
DI 1620 data 176, 3, 76, 72, 178, 169, 199, 205, 45
HH 1630 data 192, 169, 0, 237, 46, 192, 144, 241, 96
BE 1640 data 32, 77, 192, 32, 121, 193, 240, 2, 169
KH 1650 data 128, 141, 53, 192, 32, 121, 0, 240, 3
CF 1660 data 32, 30, 194, 173, 0, 221, 9, 3, 73
OI 1670 data 3, 141, 0, 221, 173, 24, 208, 41, 7
FI 1680 data 9, 8, 9, 48, 141, 24, 208, 173, 17
HJ 1690 data 208, 9, 32, 141, 17, 208, 44, 53, 192
NA 1700 data 16, 12, 173, 22, 208, 9, 16, 141, 22
LF 1710 data 208, 169, 3, 208, 10, 173, 22, 208, 41
EB 1720 data 239, 141, 22, 208, 169, 7, 141, 54, 192
OE 1730 data 73, 255, 141, 55, 192, 169, 255, 141, 51
ID 1740 data 192, 96, 169, 1, 141, 65, 192, 173, 67
PF 1750 data 192, 141, 66, 192, 169, 128, 141, 52, 192
GM 1760 data 32, 135, 193, 173, 45, 192, 10, 10, 10
ML 1770 data 10, 141, 62, 192, 141, 70, 192, 173, 43
NN 1780 data 192, 41, 15, 141, 61, 192, 44, 53, 192
HB 1790 data 48, 12, 13, 62, 192, 141, 62, 192, 141
IO 1800 data 70, 192, 76, 75, 193, 141, 33, 208, 32
OP 1810 data 121, 193, 41, 15, 141, 63, 192, 32, 121
BI 1820 data 193, 141, 64, 192, 173, 62, 192, 76, 75
DJ 1830 data 193, 32, 135, 193, 162, 3, 189, 43, 192
AJ 1840 data 157, 39, 192, 202, 16, 247, 96, 56, 169
NG 1850 data 199, 237, 41, 192, 72, 74, 74, 74, 133
PB 1860 data 252, 160, 0, 132, 251, 74, 102, 251, 74
KE 1870 data 102, 251, 101, 252, 133, 252, 173, 39, 192
JG 1880 data 174, 40, 192, 45, 55, 192, 44, 53, 192
PN 1890 data 16, 6, 10, 72, 138, 42, 170, 104, 24
DG 1900 data 101, 251, 133, 251, 138, 101, 252, 133, 252
LG 1910 data 104, 41, 7, 24, 101, 251, 133, 251, 133
DL 1920 data 253, 144, 2, 230, 252, 165, 252, 74, 102
FJ 1930 data 253, 74, 102, 253, 74, 102, 253, 133, 254
IC 1940 data 44, 53, 192, 48, 16, 24, 169, 0, 101
AL 1950 data 253, 133, 253, 169, 204, 101, 254, 133, 254
KA 1960 data 76, 249, 194, 173, 65, 192, 201, 3, 144
FC 1970 data 234, 24, 169, 0, 101, 253, 133, 253, 169
LK 1980 data 216, 101, 254, 133, 254, 24, 165, 251, 105
NG 1990 data 0, 133, 251, 165, 252, 105, 224, 133, 252
MC 2000 data 173, 39, 192, 45, 54, 192, 170, 96, 169
MO 2010 data 0, 168, 44, 52, 192, 16, 4, 112, 20
MB 2020 data 80, 15, 36, 2, 48, 9, 169, 255, 133
CP 2030 data 2, 36, 107, 48, 1, 96, 177, 251, 77
IH 2040 data 51, 192, 44, 53, 192, 48, 10, 61, 86
HF 2050 data 195, 133, 97, 189, 86, 195, 208, 8, 61
CJ 2060 data 94, 195, 133, 97, 189, 94, 195, 73, 255
CG 2070 data 49, 251, 5, 97, 145, 251, 177, 253, 45
KE 2080 data 66, 192, 13, 70, 192, 145, 253, 96, 128
KF 2090 data 64, 32, 16, 8, 4, 2, 1, 192, 48
IJ 2100 data 12, 3, 32, 110, 194, 32, 121, 0, 240
LI 2110 data 11, 32, 228, 196, 32, 121, 0, 240, 3
NK 2120 data 32, 214, 196, 32, 201, 192, 32, 125, 194
AE 2130 data 32, 14, 195, 76, 218, 192, 169, 1, 149
EH 2140 data 106, 169, 0, 149, 107, 56, 189, 43, 192
HE 2150 data 253, 39, 192, 149, 98, 189, 44, 192, 253
KN 2160 data 40, 192, 149, 99, 16, 20, 169, 255, 149
MM 2170 data 106, 149, 107, 56, 169, 0, 245, 98, 149
PJ 2180 data 98, 169, 0, 245, 99, 149, 99, 96, 21
PO 2190 data 98, 208, 4, 149, 106, 149, 107, 96, 165
DA 2200 data 99, 74, 133, 103, 165, 98, 106, 133, 102
LO 2210 data 24, 169, 0, 229, 98, 133, 104, 169, 0
```

```
ND 2220 data 229,  99, 133, 105,  96,  24, 165, 102, 101
FH 2230 data 100, 133, 102, 170, 165, 103, 101, 101, 133
LA 2240 data 103, 197,  99, 144,  19, 208,   4, 228,  98
ME 2250 data 144,  13, 138,  56, 229,  98, 133, 102, 165
OA 2260 data 103, 229,  99, 133, 103,  56,  96,  32, 135
PN 2270 data 193,  32, 121,   0, 240,  44, 201, 164, 208
BN 2280 data  16,  32, 113, 194,  32, 115,   0,  32, 138
KP 2290 data 193,  32, 121,   0, 201,  44, 208,  13,  32
HA 2300 data 228, 196,  32, 121,   0, 201,  44, 208,   3
OB 2310 data  32, 214, 196,  32,  43, 196,  32, 121,   0
MA 2320 data 201, 164, 240, 220,  96,  32, 201, 192, 162
NE 2330 data   0, 134,   2,  32, 129, 195, 162,   2,  32
OM 2340 data 129, 195, 165,  98, 197, 100, 165,  99, 229
GC 2350 data 101, 144,  62,  32, 185, 195,  36, 107,  16
BB 2360 data  10,  32, 113, 194,  56, 169,   0, 229, 108
CL 2370 data 133, 108,  32, 125, 194,  32,  14, 195, 230
BG 2380 data 104, 208,   4, 230, 105, 240, 102, 238,  39
CG 2390 data 192, 208,   3, 238,  40, 192,  32, 209, 195
AG 2400 data 144,   9,  24, 173,  41, 192, 101, 108, 141
IG 2410 data  41, 192,  32, 125, 194,  32,  14, 195,  76
MM 2420 data  91, 196, 162,   1, 181,  98, 180, 100, 149
KP 2430 data 100, 148,  98, 202,  16, 245,  32, 185, 195
EL 2440 data  36, 107,  16,  10,  32, 113, 194,  56, 169
BG 2450 data   0, 229, 108, 133, 108,  32, 125, 194,  32
OK 2460 data  14, 195, 230, 104, 240,  31,  24, 173,  41
AM 2470 data 192, 101, 108, 141,  41, 192,  32, 209, 195
AO 2480 data 144,   8, 238,  39, 192, 208,   3, 238,  40
OA 2490 data 192,  32, 125, 194.  32,  14, 195,  76, 166
JG 2500 data 196,  36, 107,  16,   3,  32,  14, 195,  32
MG 2510 data 113, 194,  76, 218, 192,  32, 121, 193,  41
JP 2520 data   3,  73,   3, 106, 106, 106, 141,  52, 192
KF 2530 data  96,  32, 121, 193,  41,   3, 240,  27,  44
PD 2540 data  53, 192,  16,  22, 141,  65, 192, 170, 189
FH 2550 data  61, 192, 141,  70, 192, 189,  66, 192, 141
HG 2560 data  66, 192, 189,   7, 197, 141,  51, 192,  96
NO 2570 data   0,  85, 170, 255,  32, 121, 193,  10,  10
OB 2580 data  10,  10, 141,  62, 192,  44,  53, 192,  48
GN 2590 data   9,  13,  61, 192, 141,  62, 192,  76,  51
CG 2600 data 197,  32, 121, 193,  41,  15, 141,  63, 192
CG 2610 data  32, 121, 193,  41,  15, 141,  64, 192, 174
BO 2620 data  65, 192, 189,  61, 192, 141,  70, 192, 189
BJ 2630 data  66, 192, 141,  66, 192,  96,  32, 110, 194
BL 2640 data  32, 135, 193, 162,   3, 189,  43, 192, 157
DD 2650 data  47, 192, 202,  16, 247,  32, 121,   0, 240
BL 2660 data  11,  32, 228, 196,  32, 121,   0, 240,   3
MJ 2670 data  32, 214, 196,  24, 173,  39, 192, 109,  47
AC 2680 data 192, 141,  43, 192, 173,  40, 192, 109,  48
LB 2690 data 192, 141,  44, 192, 173,  41, 192, 141,  45
PC 2700 data 192, 173,  42, 192, 141,  46, 192,  32, 156
CN 2710 data 193,  32,  43, 196,  56, 173,  45, 192, 237
LD 2720 data  49, 192, 141,  45, 192, 173,  46, 192, 237
PK 2730 data  50, 192, 141,  46, 192,  32, 181, 193,  32
LP 2740 data  43, 196,  56, 173,  43, 192, 237,  47, 192
HF 2750 data 141,  43, 192, 173,  44, 192, 237,  48, 192
IM 2760 data 141,  44, 192,  32,  43, 196,  24, 173,  45
AI 2770 data 192, 109,  49, 192, 141,  45, 192, 173,  46
OA 2780 data 192, 109,  50, 192, 141,  46, 192,  76,  43
HC 2790 data 196, 169,   0, 133, 251, 133, 252,  32, 241
GC 2800 data 183, 224,  40, 144,   3,  76,  72, 178, 142
PF 2810 data  73, 192,  32, 241, 183, 142,  74, 192, 138
DI 2820 data 240,  18, 224,  25, 176, 237,  24, 165, 251
EG 2830 data 105,  40, 133, 251, 144,   2, 230, 252, 202
IA 2840 data 208, 242,  24, 173,  73, 192, 101, 251, 133

FC 2850 data 251, 133, 253, 133,   3, 169,   0, 101, 252
BB 2860 data 133, 252,  24,  72, 105, 216, 133, 254, 104
NL 2870 data 105, 204, 133,   4,   6, 251,  38, 252,   6
CG 2880 data 251,  38, 252,   6, 251,  38, 252,  24, 165
DH 2890 data 252, 105, 224, 133, 252,  32, 253, 174,  32
OH 2900 data 158, 173,  32, 143, 173,  32, 166, 182, 170
AI 2910 data 160,   0, 232, 202, 208,   1,  96, 177,  34
GI 2920 data  32,  73, 198, 200,  76,  60, 198, 133, 215
KH 2930 data 138,  72, 152,  72, 165, 215,  48,  17, 201
NH 2940 data  32, 144,  28, 201,  96, 144,   4,  41, 223
BM 2950 data 208,   2,  41,  63,  76, 110, 199,  41, 127
DH 2960 data 201, 127, 208,   2, 169,  94, 201,  32, 144
HL 2970 data 125,  76, 108, 199, 201,  14, 208,   6,  32
LG 2980 data 219, 199,  76, 160, 199, 201,  17, 208,  11
MK 2990 data 162,  40,  32,  71, 199, 202, 208, 250,  76
NM 3000 data 160, 199, 201,  18, 208,   8, 169,   1, 141
KM 3010 data  75, 192,  76, 160, 199, 201,  29, 208,   6
EO 3020 data  32,  71, 199,  76, 160, 199, 162,   3,  44
BB 3030 data 162,  15, 221, 205, 198, 240,   6, 202,  16
CO 3040 data 248,  76, 160, 199, 189, 221, 198,  10,  10
EP 3050 data  10,  10, 141,  62, 192,  44,  53, 192,  48
PI 3060 data   6,  13,  61, 192, 141,  62, 192,  32,  51
BN 3070 data 197,  76, 160, 199,   5,  28,  30,  31,  16
IA 3080 data  28,  30,  31,   1,  21,  22,  23,  24,  25
EC 3090 data  26,  27,   1,   2,   5,   6,   0,   4,   7
AA 3100 data   3,   8,   9,  10,  11,  12,  13,  14,  15
ND 3110 data 201,  14, 208,   6,  32, 216, 199,  76, 160
EC 3120 data 199, 201,  17, 208,  11, 162,  40,  32,  28
EK 3130 data 199, 202, 208, 250,  76, 160, 199, 201,  18
HI 3140 data 208,   8, 169,   0, 141,  75, 192,  76, 160
OK 3150 data 199, 201,  29, 208, 143,  32,  28, 199,  76
AK 3160 data 160, 199, 165, 253, 208,   2, 198, 254, 198
DJ 3170 data 253, 165,   3, 208,   2, 198,   4, 198,   3
CA 3180 data  56, 165, 251, 233,   8, 133, 251, 165, 252
DD 3190 data 233,   0, 133, 252, 165, 251, 201,   0, 165
CK 3200 data 252, 233, 224, 176,   3,  32,  71, 199,  96
ME 3210 data 230, 253, 208,   2, 230, 254, 230,   3, 208
IP 3220 data   2, 230,   4,  24, 169,   8, 101, 251, 133
EM 3230 data 251, 144,   2, 230, 252, 165, 251, 201,  64
JN 3240 data 165, 252, 233, 255, 144,   3,  32,  28, 199
KB 3250 data  96,   9,  64, 174,  75, 192, 240,   2,   9
NC 3260 data 128,  32, 165, 199, 160,   7,  32, 230, 199
NM 3270 data 177,   5, 145, 251, 136,  16, 249,  32, 245
LP 3280 data 199, 200, 173,  61, 192,  44,  53, 192,  16
FO 3290 data   8, 173,  64, 192, 145, 253, 173,  63, 192
MO 3300 data  13,  62, 192, 145,   3,  32,  71, 199, 104
IP 3310 data 168, 104, 170,  96, 133,   5, 169,   0, 133
MD 3320 data   6,   6,   5,  38,   6,   6,   5,  38,   6
GB 3330 data   6,   5,  38,   6,  24, 173,  71, 192, 101
BJ 3340 data   5, 133,   5, 173,  72, 192, 101,   6, 133
MG 3350 data   6,  96,  32, 253, 174,  32, 138, 173,  32
EF 3360 data 247, 183, 166,  21, 208,   9, 165,  20, 208
AC 3370 data   3, 162, 208,  44, 162, 216, 142,  72, 192
FA 3380 data 162,   0, 142,  71, 192,  96, 173,  14, 220
JA 3390 data  41, 254, 141,  14, 220, 165,   1,  41, 251
DM 3400 data 133,   1,  96, 165,   1,   9,   4, 133,   1
EM 3410 data 173,  14, 220,   9,   1, 141,  14, 220,  96
HF 3420 data  32, 121,   0, 240,  15,  32, 110, 192,  32
LL 3430 data 121, 193, 141, 245, 192, 142, 249, 192, 169
JE 3440 data 128,  44, 169,   0, 141,  76, 192,  96,   0
AB 3450 data   0
```

Garbage Collector Revealed

Michael T. Graham
Hopatcong, New Jersey

. . .the system appears to be dead as a doornail. . .

That Sinking Feeling

You only have a few records left to enter, then you'll be done. You finish the current entry, hit RETURN, and. . . nothing! The screen doesn't clear, the cursor doesn't blink. Beads of sweat break on your forehead as the thought of all your work going down the tubes explodes on your tired brain. Seconds turn to minutes as your mind races, trying to come up with a way out of this dilemma. Finally, you push the RUN/STOP key. Still nothing! What could have happened? Has your poor old 64 given up the ghost? Then, as your other hand reaches for the RESTORE key, the system springs back to life. You finish the last few entries and examine the results. Everything is OK! After storing your work to disk, you stare at the screen and mull over the doubts you now have about the reliability of your system.

The above moment of fright was brought to you by a routine in BASIC called the garbage collector. As strings are stored and changed in BASIC, the string storage space in memory fills up with old, useless information, known in technical circles as garbage. When memory fills up with this junk, BASIC runs the garbage collector to clear it out. In a program with large string arrays, this process can take many minutes to complete and while it is running the system appears to be dead as a doornail. To add insult to injury, BASIC makes no attempt to tell you to "please stand by". How many times has the RESTORE key been hit because of this? I shudder at the thought.

Fighting Back

Garbage Collector Revealed provides a solution to this problem. When installed, this program will display a "system busy" message on the top line of the screen when the garbage collector starts to execute. When collection is finished, the original contents of the top line are restored.

This is accomplished by adding a few patches to BASIC. The usual method of hooking into BASIC via RAM vectors won't work here, there isn't a vector that points to the garbage collector. Instead, the BASIC ROM is copied to RAM and then switched out. The entry and exit points of the garbage collector are then patched to call small routines to display and remove the busy message. This all happens in a flash when the message program is initialized. When initialization is complete the system is executing BASIC in RAM, patches and all.

Now when BASIC tries to run the garbage collector, one of the patches installed above steers execution to the routine that displays the busy message. This routine first stores the existing contents of the screen and color memory locations used by the message. The message is then displayed in the color of your choice. When this process is complete, control is passed to the garbage collector.

When the garbage collector is finished, the other patch executes the routine that restores the screen to its original contents. This is accomplished by simply storing the saved color and character bytes back where they came from. The screen is now exactly as it was when this all started. Control is then returned to BASIC.

How To Use It

First type and save program 1. This program will create a machine language program file on disk called GARBMSG containing the message program code. This file loads the message program to the cassette buffer from memory locations 828 to 1019. Memory locations 2, 251 through 255, and 679 through 718 are used for working storage.

When Program 1 is RUN, it will prompt you to insert the disk that GARBMSG is to be created on. Do so and hit the RETURN key. The program will then be placed on this disk. The disk drive may start and stop several times during the process, wait for the program to end before removing the disk.

To use the message patch in your BASIC application, insert the following as the first two executable lines in the program:

```
10 IF A=0 THEN A=1:LOAD "GARBMSG",8,1
20 CLR:SYS 828:POKE 2,C
```

The C in line 20 should be replaced with the number of the color you want the message text to be displayed in. If the POKE in line 20 is left out, the message will be displayed in white. The message can be displayed in reverse text by adding the following line:

```
30 FOR I=1000 TO 1019:POKE I,(PEEK(I) OR 128):NEXT I
```

If your BASIC program moves the location of the text screen from its normal position in memory, be sure it also updates the pointer at memory location 688 to the new screen starting page number. The Kernal uses this pointer to find the text screen and so does GARBMSG. Under normal conditions this is not required.

Note that if the system is reset with RUN/STOP-RESTORE, the message patch will be disconnected. running your BASIC program again will re-install it, so this shouldn't be a problem.

As is, the garbage collector will run whenever BASIC decides that memory is getting short. You can force garbage collection to occur at any point in your program by inserting X=FRE(0) at an appropriate line in your program. The FRE( ) function runs the garbage collector so it can give you a true indication of your remaining memory space. You can use this to get garbage collection "over with" at a strategic spot in your program, such as when other lengthy processes are taking place. In either case, the message will be displayed while the garbage collector is running.

To test the message program, enter program 2. This program is designed to do nothing but create garbage. Be sure to have a disk containing GARBMSG in the drive when you RUN it. If all is well, you should see the message being displayed every few seconds. Note that the original text and color is restored when the message disappears. Note also the use of a contrasting color and reversed text for the message, this will make it easier to see on a crowded screen. Lines 160 to 180 demonstrate the use of the FRE( ) function to force a run of the garbage collector.

Although there is no easy way to rid yourself of the infamous garbage collector, at least now you will know when it runs. You can now use the time you worried through for more productive purposes, such as getting a fresh beer or watching a few minutes of the Brady Bunch.

Happy Computing!

GarbMessage: Demonstration Program

```
MM 100 rem garbage collector message demo
DB 110 rem by mike graham  12/02/86
DA 120 if a=0then a=1:load "garbmsg",8,1
MI 130 clr:sys 828:poke 2,7:rem color=yellow
FK 140 for i=1000 to 1019
MG 150 poke i,(peek(i) or 128):rem reverse garb
       message text
PH 160 next i
FG 170 poke 56,20:clr:rem restrict memory size
       (normally 160)
KB 180 t$="garbage"
OD 190 print" S   a line of text at the top of the screen."
MC 200 dim a$(200)
FJ 210 for i=1 to 7
CI 220 for j=0 to 200
NM 230 a$(j)=left$(t$,i)
CK 240 print" sqqq    "j"[3 spcs]";
MN 250 next j
LC 260 print tab(9)a$(1)"[6 spcs]"
NO 270 next i
AJ 280 print" sqqqqq   forced collection"
GA 290 x=fre(0):rem force garbage collection
PA 300 print" sqqqqq   [17 spcs]"
CN 310 goto210
```

GarbMessage: Creates ML on disk

```
IF 100 rem generates gargage collector message
DB 110 rem by mike graham 12/02/86
LK 120 for j=1 to 192 : read x
HB 130 ch=ch+x : next
LE 140 if ch<>22794 then print "checksum error"
       : end
LE 150 print "data ok, now creating file" : print
IL 160 restore
ID 170 open8,8,8,"0:garbmsg,p,w"
HO 180 print#8,chr$(60)chr$(3); :rem start addr=828
       ($033c)
BP 190 for j=1 to 192 : read x
IE 200 print#8,chr$(x); : next
KL 210 close 8
OF 220 print "prg file 'garbmsg' created. . .
AO 230 print "this generator no longer needed.
OA 240 rem
CO 250 data 169,   0, 133, 251, 169, 160, 133, 252
DO 260 data 160,   0, 177, 251, 145, 251, 200, 208
BD 270 data 249, 230, 252, 165, 252, 201, 192,  48
EF 280 data 241, 169,  76, 141,  38, 181, 141,   6
FB 290 data 182, 169, 128, 141,  39, 181, 169,   3
LP 300 data 141,  40, 181, 169, 138, 141,   7, 182
LG 310 data 169,   3, 141,   8, 182, 169, 239, 133
LM 320 data   0, 165,   1,  41, 254, 133,   1, 169
IG 330 data   1, 133,   2,  96,  32, 153,   3, 166
HL 340 data  55, 165,  56,  76,  42, 181, 165,  79
BH 350 data   5,  78, 240,   3,  76,  12, 182,  32
GL 360 data 198,   3,  76,   1, 182,   8, 152,  72
EC 370 data 169,  10, 133, 253, 173, 136,   2, 133
BD 380 data 254, 160,  19, 166,   2, 177, 253, 153
AI 390 data 167,   2, 185, 232,   3,  41, 191, 145
PF 400 data 253, 185,  10, 216, 153, 187,   2, 138
ML 410 data 153,  10, 216, 136,  16, 231, 104, 168
BF 420 data  40,  96,   8,  72, 152,  72, 169,  10
FL 430 data 133, 253, 173, 136,   2, 133, 254, 160
AI 440 data  19, 185, 167,   2, 145, 253, 185, 187
AK 450 data   2, 153,  10, 216, 136,  16, 242, 104
AD 460 data 168, 104,  40,  96,  32,  62,  87,  65
GG 470 data  73,  84,  44,  83,  89,  83,  84,  69
FF 480 data  77,  32,  66,  85,  83,  89,  60,  32
```

GarbMessage: PAL Source Code

```
PH 1000 rem --- garbage collector message ---
CO 1010 rem ------ by michael t. graham ------
IH 1020 rem open8,8,8,"0:garbmsg,p,w"
HN 1030 sys700
BB 1040 ;.opt o8
BC 1050 ;pal assembler source code
KJ 1060 ;
PC 1070 ;this basic patch puts a message
FD 1080 ;on screen when basic's garbage
CP 1090 ;collector runs. basic is moved to
NP 1100 ;ram at initialization.
MM 1110 ;
DB 1120 hibase = $0288   ;kernal screen page addr
EF 1130 addr   = $fb     ;move vector
```

```
MC 1140 patch1 = $b526  ;patch start of inputor
MM 1150 patch2 = $b606  ;patch end of inputor
LO 1160 ret1   = $b52a  ;continue start
IH 1170 cont   = $b60c  ;inputor not done
DO 1180 quit   = $b601  ;end inpution
EB 1190 color  = $02    ;message color storage
BK 1200 cmem   = $d80a  ;color memory start
NG 1210 screen = $fd    ;screen vector storage
KD 1220 ;
GK 1230 *=$033c               ;cassette buffer
OE 1240 ;
FG 1250 ;initialize message patch (sys 828)
CG 1260 ;
CA 1270 init   lda  #$00      ;set up
GG 1280        sta  addr      ;indirect
KB 1290        lda  #$a0      ;address
FO 1300        sta  addr+1    ;to move
OD 1310        ldy  #$00      ;basic
KM 1320 xfer   lda  (addr),y  ;move basic rom
EL 1330        sta  (addr),y  ;to ram
MB 1340        iny
DM 1350        bne  xfer
KA 1360        inc  addr+1    ;bump page address
NL 1370        lda  addr+1    ;continue move
KG 1380        cmp  #$c0      ; up to
LC 1390        bmi  xfer      ;$bfff
FI 1400        lda  #$4c      ;jmp instruction op code
BL 1410        sta  patch1    ;store the jumps for the
GG 1420        sta  patch2    ;patches
GD 1430        lda  #<fix1    ;jump to fix1 at start
MI 1440        sta  patch1+1
HE 1450        lda  #>fix1
EK 1460        sta  patch1+2
KL 1470        lda  #<fix2    ;jump to fix2 at end
GL 1480        sta  patch2+1
BH 1490        lda  #>fix2
OM 1500        sta  patch2+2
IH 1510        lda  #$ef      ;map
LI 1520        sta  $00       ;out
OO 1530        lda  $01       ;the basic
NA 1540        and  #$fe      ;rom
LM 1550        sta  $01
JE 1560        lda  #01       ;initialize
IA 1570        sta  color     ;color to white
HF 1580        rts            ;end init
MK 1590 ;
KN 1600 ;patch handler
AM 1610 ;
MB 1620 ;fix1 = patch to start of inputor
EN 1630 ;
EH 1640 fix1   jsr  msgon     ;display message
EF 1650        ldx  $37       ;finish displaced basic
DM 1660        lda  $38       ;code
GP 1670        jmp  ret1      ;return to basic
GA 1680 ;
EM 1690 ;fix2 = patch to end of inputor
KB 1700 ;
JI 1710 fix2   lda  $4f       ;finish basic's
OA 1720        ora  $4e       ;stuff
DF 1730        beq  retrn     ;done inpution
PP 1740        jmp  cont      ;continue inpution
DC 1750 retrn  jsr  msgoff    ;put screen back
JN 1760        jmp  quit      ;inputor done
AG 1770 ;
```

```
KB 1780 ;save user screen and put message
BC 1790 ;on screen before starting
LL 1800 ;garbage inputor.
II 1810 ;
BM 1820 msgon php             ;save status
BI 1830        tya            ;and y
HM 1840        pha            ;register
LH 1850        lda  #10       ;start at
AK 1860        sta  screen    ;11th char
AD 1870        lda  hibase    ;get screen
DD 1880        sta  screen+1  ;page addr
FJ 1890        ldy  #19       ;20 characters
IO 1900        ldx  color     ;message color
FC 1910 move   lda  (screen),y ;save existing
JA 1920        sta  buffer,y  ;screen contents
KF 1930        lda  messag,y  ;get message
PA 1940        and  #$bf      ;convert to screen code
JD 1950        sta  (screen),y ;display message
HN 1960        lda  cmem,y    ;save existing
HC 1970        sta  cbuff,y   ;color memory
OA 1980        txa            ;replace with
JH 1990        sta  cmem,y    ;message color
OE 2000        dey            ;bump index
HP 2010        bpl  move      ;move 20 characters
PL 2020        pla            ;restore
OH 2030        tay            ;registers
ML 2040        plp
IH 2050        rts            ;done
CI 2060 ;
AL 2070 ;put user's screen contents back
FK 2080 ;after collector is done.
AK 2090 ;
HF 2100 msgoff php            ;save
KL 2110        pha
AM 2120        tya            ;registers
OM 2130        pha
LH 2140        lda  #10       ;11th character
JI 2150        sta  screen    ;on screen
CD 2160        lda  hibase    ;screen page
AG 2170        sta  screen+1  ;address
HL 2180        ldy  #19       ;20 characters
NG 2190 restor lda  buffer,y  ;put text
JL 2200        sta  (screen),y ;back
JA 2210        lda  cbuff,y   ;restore
AB 2220        sta  cmem,y    ;colors
ED 2230        dey            ;bump index
OF 2240        bpl  restor    ;restore 20 characters
FK 2250        pla            ;restore
DK 2260        tay
HC 2270        pla            ;registers
MK 2280        plp
IG 2290        rts            ;done
CH 2300 ;
EE 2310 messag  .asc "  >wait,system busy<  "
GI 2320 ;
LN 2330 ;data storage - 40 bytes at $02a7
KJ 2340 ;
PF 2350 *=$02a7
OK 2360 ;
DF 2370 buffer  *=*+20
AO 2380 cbuff   *=*+20
MM 2390 ;
MD 2400 .end
```

SYS 65478: Taking A New Look at an Old Dog

Miklos Garamszeghy
Toronto, ON

. . .the KERNAL CHKIN routine can be put to some much less obvious uses. . .

SYS 65478 (that's JSR $FFC6 to you machine language fans) seems like an innocent and well documented routine. Opening a file as an input channel. What could be simpler? While this may be its normal use from within a program, the KERNAL CHKIN routine can be put to some much less obvious uses in immediate mode. One of the more interesting of these enables a Commodore 64 or 128 (and other CBM machines with a similar KERNAL structure) to execute a series of commands contained in a disk file, similar to an MS–DOS batch file or a CP/M submit file.

The CHKIN routine resets the input device flag (normally 0 to indicate the keyboard) at zero page location hex $99 (on the VIC–20, C–64 and C–128), $AF (on the BASIC 2.0/4.0 PETS), $A1 (on the B series) or $98 (on the Plus/4 and C–16) to a value corresponding to the device from which normal input would be received. The main BASIC immediate mode input loop checks this location before trying to fetch an input byte. If the value is 0, a normal entry occurs by fetching a byte from the keyboard buffer. However, if the value is 8, for example, the fetch routine will attempt to read a byte from serial port device 8 (usually a disk drive). If device 8 has an open file capable of giving output, a byte is read from this file and placed in BASIC's input buffer, just as if it had been entered from the keyboard. If a carriage return is encountered, the commands contained in the input buffer are executed, assuming there is no line number at the beginning. Thus you can execute commands contained in a disk file. It should be noted that even with normal input redirected to respond to an external device, the keyboard is still scanned by the IRQ routines and the results placed in the keyboard buffer only, up to the maximum number of characters allowed for the buffer. BASIC's input editor will not see any of these characters until control has been restored to the keyboard.

So what, you say? Everyone knows an easier way to do that: It's called a program or PRG file. Let's make one thing perfectly clear. Executing what I call a sequential disk command (SDC) file is not the same thing as LOADing and RUNning a disk PRG program file. There are several major differences: A regular PRG type file contains a series of tokenized BASIC lines (complete with link addresses and line numbers) or machine language op codes. The sequential disk command file, on the other hand, contains a series of immediate mode commands written out in plain English, just as you would type them in from the keyboard. In addition, a PRG file must be resident in RAM for execution. In most 8 bit computer operating systems (Commodore KERNALs included), only one program can be in memory for execution at a given time (assuming that you have not artificially partitioned the memory into separate work spaces.) An SDC–type program is not memory resident! It resides entirely in a disk file and is called up and executed statement by statement, without affecting any program(s) stored in the computer's main RAM unless you deliberately want to. However, since it resides on disk, an SDC program usually takes longer to execute than a RAM resident program because of Commodore's notoriously slow disk access speeds.

SDCs are useful as utility and easy to use reference data files since they can be called up and executed without fear of erasing main computer memory. This allows a programmer to interrupt work, call up and consult an on line data table, for example, and then resume the task at hand, all with relative ease and speed. SDCs can also be used for storing customized keyboard macros. (For those unfamiliar with the concept, a keyboard macro is an often lengthy and/or complicated series of frequently used keystrokes/commands that can be automatically invoked by using a shorter, easier to remember key sequence.) These can be very useful for setting up sprites, sound and graphics on older CBM machines such as the C–64 that lack high level commands for doing so. (Can you honestly remember the POKE sequence to play the first few bars of HAPPY BIRTHDAY on the C–64?)

Setting up an SDC

A sequential disk command file is very easy to create. You use your favourite word processing program to create a SEQ file containing a series of immediate mode BASIC commands, just as you would type them in to execute from the keyboard. BASIC keywords (such as PRINT) can be entered in either their long or abbreviated forms. Of course, the same limits on line length as for normal programming apply to the lines in your SDC file (e.g. 80 characters for the C–64, 160 characters for the C–128, etc). This is a restriction imposed by the size of the input buffer on the computer.

Any immediate mode command can be used except for disk access commands. You should not use OPEN, LOAD, SAVE, DIRECTORY, etc. because these commands will reset the input device to the default keyboard value after they have executed, thus cutting off the rest of your command file. A DIRECTORY can be used as the last item in an SDC because it will return control to the keyboard upon execution, which is desirable in this case.

The program mode only commands, such as INPUT, GET, GOTO, GOSUB, etc., cannot be used in SDCs because an SDC is executed in immediate mode, not under program control.

In order to be properly interpreted when they are read in, the BASIC keywords must be typed in a style that allows them to be saved as un–shifted PETSCII characters in a disk file. (This is the way that

you would normally enter them from the keyboard.) With most word processors, such as PaperClip or Pocket Writer–128, this means that they are typed in lower case only and the file is saved as a SEQ file. With word processors such as Timework's Word Writer 128, which save text in true ASCII format, the BASIC keywords must be entered in UPPER CASE only. Word processors that use PRG type screen code files only should not be used for creating SDCs.

Making a graceful exit from an SDC back to keyboard input can be somewhat tricky. The easiest way is to include a statement at the end of your SDC which POKEs a 0 value back into the input device flag location described earlier. Simply CLOSEing the disk file from within the SDC or using the BASIC commands END or STOP, will not return control to the keyboard because they do not automatically reset the input device flag. Another way to exit is to include a garbage statement or deliberate syntax error as the last line. Upon reaching such an error, the SDC will crash and control will be returned to the keyboard. The least elegant way to exit is by the familiar <RUN–STOP>/<RESTORE> key combination. Crude but effective.

Once the SDC has been entered, it should be saved as a SEQ disk file with an appropriate name.

Executing the SDC

Now comes the fun part. Executing the SDC is really quite simple. All that is required is to open the disk file in immediate mode with a statement such as:

```
OPEN 1,8,8,"filename"
```

Second, you must activate the CHKIN routine. On the C–128, with the above OPEN statement you would use:

```
SYS 65478,0,1
```

With a C–64 or similar type machine, a double statement is required:

```
POKE 781,1
```

followed by:

```
SYS 65478
```

The commands in the SDC will then be executed, one line at a time until control is returned to the keyboard by one of the methods previously outlined. You will note that the actual commands are not printed to the screen before they are executed, but the "READY." message is printed after each line has been executed. Once the SDC has finished executing, the disk file should be closed with a DCLOSE or CLOSExx as applicable for your machine.

SDCs on the C–128

The Kernal input routine will accept line numbers in SDCs. These line numbered files will "execute" exactly as if the line, complete with line number, had been entered from the keyboard. That is, it will be added to any program currently in memory. This little trick is a simple yet powerful way to MERGE two or more program files on the C–128. Unlike other pseudo merge routines which merely append one program to the end of another, this technique allows full intermeshing of line numbers. Only a few steps are required. First, LOAD one of the programs into memory in the normal manner. Next convert it into a SEQ disk file listing with a series of commands such as:

```
OPEN 8,8,8,"0:filename,S,W":CMD8:LIST
PRINT#8:CLOSE8
```

LOAD in the second file. With the second file now in memory, activate the SEQ listing of the first file as a SDC as outlined above. The programs will now be MERGEd. The merge will terminate with a mysterious "OUT OF DATA" error message. This is a good sign: the process worked. The error message is caused by the last line of the listing in the disk file – "READY.". (Commodore BASIC listings always include this.) The computer, not being able to recognize its own writing, thinks that someone typed in "READ Y". Since there are no accompanying DATA statements, the out of data error occurs and control is restored to the keyboard. If the READY. message did not appear at the end of the listing (for example if you edited it out with a word processor to give it a neater appearance), keyboard control would only be restored with a <RUN–STOP>/<RESTORE>. The programs will, however, be merged correctly. This technique can also be used for "loading" program listings produced on machines with incompatible keyword tokens and programs transferred into SEQ files via a modem download.

Example SDCs

LISTINGs 1, 2 and 3 are short examples of SDCs. While it may appear that some of the statements are repetitive, it should be remembered that they were created with a word processor. (If your word processor does not have cut, paste and copy commands, then perhaps it is time to splurge for a new one!)

LISTING 1 is an example for the C–64 or C–128 (in 40 or 80 column mode) that prints out a simple calendar for the month of April, 1987. You will note that most of the lines begin with the sequence "print up$". "up$" is defined in the first line as three cursor ups. This allows you to get around the nasty habit of immediate mode BASIC of printing a few carriage returns and a READY after each line it executes. up$ is included to properly format the screen display in this case. Note also that special control characters are given as their CHR$() values. This is the only way to enter them with a word processor.

LISTING 2 will print out a handy hex–dec conversion chart on the 80 column screen of the C–128. It is similar in nature to the above example, but works in C–128 FAST mode. The 80 column screen is required because of the width of the table.

The final example, LISTING 3, is interesting for several reasons. It is essentially a self–contained data file that can read and display itself on the screen automatically! The WAIT and POKE values of 208 in the lines are set up for a C–128. For a C–64, change all of the 208s to 198s. This is the location of the keyboard buffer flag that indicates the number of characters in the buffer. The file will display one entry at a time and wait for you to press a key before displaying the next entry. (Remember, the keyboard scan and buffer filling still

occurs when an SDC is being executed even if its input is ignored by the BASIC input routine, hence the need to clear the buffer by POKEing a 0 to it before reading the next entry.) To stop the display before it reaches the end, a <RUN-STOP>/<RESTORE> should be used. This last example demonstrates that, although you cannot read the keyboard directly with GETs, INPUTs, etc., you can still obtain data from the keyboard via direct PEEKs and POKEs to the keyboard buffer areas.

Variations on a Theme

The re-directed input is not limited to disk files. The procedure works equally well with the user port. You could, in theory, control your computer in immediate mode from a remote location with an external keyboard or even a modem and an auxiliary terminal.

Listing 1: for C-64 or C-128

```
up$=chr$(145)+chr$(145)+chr$(145):poke53280,6:poke53281,6
:printchr$(147)chr$(5);print up$"[13 spcs]august 1987"
print up$"   sun   mon   tues   wed   thur   fri   sat   "
print up$"+----+----+----+----+----+----+----+"
print up$"                                    1   "
print up$"+----+----+----+----+----+----+----+"
print up$"    2     3     4     5     6     7     8   "
print up$"+----+----+----+----+----+----+----+"
print up$"    9    10    11    12    13    14    15   "
print up$"+----+----+----+----+----+----+----+"
print up$"   16    17    18    19    20    21    22   "
print up$"+----+----+----+----+----+----+----+"
print up$"   23    24    25    26    27    28    29   "
print up$"+----+----+----+----+----+----+----+"
print up$"   30    31                                 "
print up$"+----+----+----+----+----+----+----+"
poke 153,0
```

Listing 2: for C-128 (80 column only)

```
up$=chr$(145)+chr$(145)+chr$(145):printchr$(147)chr$(5);:fast
print up$"[9 spcs]hex-dec converter"
print up$"       00  01  02  03  04  05  06  07  08  09  0a  0b  0c  0d  0e  0f  "
print up$"+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+"
print up$"  00 :   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  "
print up$"  10 :  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  "
print up$"  20 :  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  "
print up$"  30 :  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  "
print up$"  40 :  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  "
print up$"  50 :  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  "
print up$"  60 :  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111  "
print up$"  70 : 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127  "
print up$"  80 : 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143  "
print up$"  90 : 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159  "
print up$"  a0 : 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175  "
print up$"  b0 : 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191  "
print up$"  c0 : 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207  "
print up$"  d0 : 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223  "
print up$"  e0 : 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239  "
print up$"  f0 : 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255  "
print up$:slow:poke 153,0
```

Listing 3: for C-128 (for C-64, change all 208s to 198)

```
up$=chr$(145)+chr$(145):printchr$(147)chr$(5);
print up$"              mini phone file ":print
print up$" name: john doe     phone:123-456-7890":wait 208,1:poke208,0
print up$" name: mary brown   phone:123-456-7890":wait 208,1:poke208,0
print up$" name: jim smith    phone:123-456-7890":wait 208,1:poke208,0
print up$" name: jane green   phone:123-456-7890":wait 208,1:poke208,0
print up$" name: bill black   phone:123-456-7890":wait 208,1:poke208,0
print up$" name: fred right   phone:123-456-7890":wait 208,1:poke208,0
print up$" name: sally ho     phone:123-456-7890":wait 208,1:poke208,0
poke153,0
```

Kernal LISTEN And Its Relatives

Eric Germain
Ste–Foy, Quebec

In Volume 6, Issue 05 of The Transactor, an article, 'Assembly Language Disk Access', written by Richard Evers, was published. This article talks about the techniques used to communicate through the serial bus. The methods presented make use of some Kernal routines: OPEN, CLOSE, CHRIN and CHROUT, among others. The use of these techniques offer many advantages. First, you don't have to worry about the various parameters needed to communicate; you just have to specify the device number, the secondary address, the logical file number and the filename, while the computer calls the specified device by sending individual bit patterns on the bus and getting ones from the device.

The second advantage is that once you have opened a file, its handling is almost executed the same way as in BASIC. Thus, you just have to remember its logical file number. Everything else is kept in tables in RAM which are automatically handled by the Kernal.

Third, outputing a character is very simple; you put its ASCII code in the accumulator and call CHROUT. Getting a character from the bus is even more simple since you just have to call CHRIN.

There are, however, a few drawbacks using these methods. While writing some TransBasic modules of my own, I have experienced associated problems.

Let's imagine you have written a TransBasic VALIDATE command which may look like one of these:

```
start   lda  #8        ;set logical,
        tax             ;device
        ldy  #15        ;and secondary address
        jsr  $ffba      ;kernal setlfs
        lda  #0         ;no file name
        jsr  $ffbd      ;kernal setnam
        jsr  $ffc0      ;kernal open
        ldx  #8         ;logical file #
        jsr  $ffc9      ;kernal chkout
        lda  #"v"       ;validate command
        jsr  $ffd2      ;kernal chrout
        jsr  $ffcc      ;kernal clrchn
        lda  #8         ;logical file #
        jsr  $ffc3      ;kernal close
        rts             ;return to main program
```

- - - - -

```
start   lda  #8         ;set logical,
        tax             ;device
        ldy  #15        ;and secondary address
        jsr  $ffba      ;kernal setlfs
        lda  #1         ;set name length
        ldx  #<name     ;address lo
        ldy  #>name     ;address hi
        jsr  $ffbd      ;kernal setnam
        jsr  $ffc0      ;kernal open
        lda  #8         ;logical file #
        jsr  $ffc3      ;kernal close
        rts             ;return to main program
name    .asc "v"        ;validate command
```

Both listings use the usual machine language method of OPENing a file, printing to it via CHROUT, and CLOSEing it. When sending to the disk drive a slow–execution command, like VALIDATE, the method has a major drawback: the CLOSE command is only executed when the drive has finished its work. In other words, control is not returned to the main program (in the case of a TransBasic command, it's the BASIC itself) until the VALIDATE has been completed !

Also, both listings are about twice as long as they could be. The second one is especially deceiving if you are using a mini–assembler such as Supermon, since you must find yourself NAME's address.

The Alternative

There are many ways to get around these problems very easily. But first, let's see how do some Kernal routines work.

The LISTEN routine is used to call a device on the serial bus; it sends a LISTEN signal on the bus, along with the number of the device to communicate with. The SECOND routine works together with LISTEN; it sends a SECOND signal along with a command byte. The low nybble of the byte is just the secondary address of the device; bits 5 and 6 are always on; bits 4 and 7 are special control bits.

The TALK and TKSA routines work exactly the same as LISTEN and SECOND. The only differences are in the signals sent through the bus. When a device receives LISTEN and SECOND signals, it prepares to receive data from the bus; when it receives TALK and TKSA signals, it prepares to send some data.

The OPEN routine works by just calling LISTEN and SECOND: the command byte has control bits 4 and 7 set to indicate that a file is to be OPENed. Following the command byte is the filename, if present, sent one character at a time. Then comes an UNLISTEN signal (obtained by calling the Kernal routine UNLSN).

CLOSE works the same way with the difference that no filename is sent and that the control bit 4 in the command byte is set OFF (bit 7 remains ON).

It appears that OPENing a file is just necessary to tell the device the filename you want to use. In the case of the error channel, that one used to send VALIDATE commands, there is no filename: you don't need to open the channel! And since it's not open, you don't have to close it! We can save two major steps by writing our own routines using only LISTEN and a few other routines. Try the following:

```
start   lda  #8         ;device number
        jsr  $ffb1      ;kernal listen
        lda  #$6f       ;secondary address or $60
        jsr  $ff93      ;kernal second
        lda  #"v"       ;validate command
        jsr  $ffa8      ;kernal ciout
        jsr  $ffae      ;kernal unlsn
        rts             ;return to main program
```

After setting the device number, we send it through the serial bus by calling LISTEN; then, we set the secondary address OR'ed with $60 (to set bits 5 and 6 which must ALWAYS be on) and send it with SECOND. These first four instructions do the same job as CHKOUT routine, except that you don't have to open a useless file. Short and sweet.

CIOUT works in a very handy way: it will send information on the bus to whatever is listening! We can use this feature to perform some very convenient data transfers. We could, for instance, tell two disk drives, or one drive and a printer, to LISTEN simultaneously, and then send some data which will be received by BOTH devices. Note that two devices, including the main computer, cannot TALK at the same time; collision between data would cause a bus crash.

Communicating with the printer and the command/error channel of the disk drive is probably the biggest advantage of the technique described above. Communicating with a disk file would, however, be a little tedious. It may sometimes be more advised to call the OPEN routine rather than sending the filename one character at a time. The choice is yours.

I have included addresses of the nine major Kernal routines described in the article:

ACPTR	$FFA5	CIOUT	$FFA8
LISTEN	$FFB1	READST	$FFB7
SECOND	$FF93	TALK	$FFB4
TKSA	$FF96	UNLSN	$FFAE
UNTLK	$FFAB		

I have included READST in the list because every program should use it to check for errors. By the way, before sending any TALK signal, you must set the status byte to zero by storing a zero in location $90 (decimal 144).

Listing 1 contains the BASIC loader for the DIRECTORY program. Listing 2 contains the commented PAL source listing. Type SYS49152 to see on the screen what you would see by doing LOAD "$",8 and LIST. The routine uses all the concepts described in the article, plus some BASIC ROM routines to print block counts. You can stop it by pressing the STOP key.

I hope this article will help you to improve your programs dealing with the disk drive and the printer. Happy programming!

Listing 1: BASIC Loader

```
DG 1000 rem save"0:directory 64.ldr",8
EH 1010 rem * directory demo for the c64
FM 1020 rem * sys(49152) to activate
KH 1030 :
BL 1040 for j=49152 to 49262: read x: poke j,x
       : ch=ch+x: next
AA 1050 if ch<>15837 then print "checksum error"
       : stop
IJ 1060 :
JE 1070 data 169,  8,  32, 177, 255, 169, 240,  32
EJ 1080 data 147, 255, 169,  36,  32, 168, 255, 169
FN 1090 data  48,  32, 168, 255,  32, 174, 255, 169
AE 1100 data   8,  32, 180, 255, 169,  96,  32, 150
OL 1110 data 255, 169,   0, 133, 144,  32, 165, 255
LN 1120 data  32, 165, 255,  32, 211, 170,  32, 165
DM 1130 data 255,  32, 165, 255,  32, 165, 255, 168
AF 1140 data  32, 165, 255, 166, 144, 208,  31,  32
KA 1150 data 145, 179, 160,   1,  32, 215, 189, 169
EF 1160 data  32,  32, 210, 255,  32, 165, 255,   8
PP 1170 data  32, 210, 255,  40, 208, 246,  32, 211
BA 1180 data 170,  32, 225, 255, 208, 208,  32, 171
MD 1190 data 255, 169,   8,  32, 177, 255, 169, 224
DH 1200 data  32, 147, 255,  32, 174, 255,  96
```

Listing 2: PAL Source Code

```
KE 1000 rem save"0:directory 64.pal",8
HB 1010 rem * by eric germain  ste-foy, quebec
JF 1020 open 8,8,1,"0:directory 64.obj"
HN 1030 sys700
DD 1040 .opt o8
LA 1050 *=$c000
KJ 1060 ;
JA 1070 status   =  $90     ;file status variable
JJ 1080 return   =  $aad3   ;send chr$(13)
CB 1090 fixflt   =  $b391   ;fixed-float conversion
EH 1100 shwflt   =  $bdd7   ;print floating point value
IM 1110 acptr    =  $ffa5   ;input byte from serial port
PG 1120 ciout    =  $ffa8   ;output byte to serial port
ME 1130 listen   =  $ffb1   ;set listen
LK 1140 second   =  $ff93   ;send sa after listen
BI 1150 talk     =  $ffb4   ;set talk
BD 1160 tksa     =  $ff96   ;send sa after talk
MB 1170 unlsn    =  $ffae   ;command bus to unlisten
EA 1180 untlk    =  $ffab   ;command bus to untalk
OM 1190 chrout   =  $ffd2   ;output character
BL 1200 stop     =  $ffe1   ;test (stop) key
AD 1210 ;
AH 1220 ;** directory read demo **
EE 1230 ;
CE 1240         lda  #8     ;device number
ON 1250         jsr  listen ;and secondary address
EB 1260         lda  #$f0   ;'or'ed with $f0
IA 1270         jsr  second ;to indicate a file to be opened
AJ 1280         lda  #"$"   ;"$0"
NM 1290         jsr  ciout  ;sent as filename
HK 1300         lda  #"0"   ;one character at a time
NF 1310         jsr  ciout
JD 1320         jsr  unlsn  ;stop listening
MJ 1330         lda  #8     ;device number
PP 1340         jsr  talk   ;and secondary address
OP 1350         lda  #$60   ;'or'ed with $60
DI 1360         jsr  tksa   ;to indicate normal i/o operation
PC 1370         lda  #0     ;set status word
KG 1380         sta  status ;to zero
DI 1390         jsr  acptr  ;get two dummies
GK 1400         jsr  acptr
HA 1410         jsr  return ;print carriage return
CA 1420 ;
NM 1430 main    =  *
FL 1440         jsr  acptr  ;get two dummies
IN 1450         jsr  acptr
FD 1460         jsr  acptr  ;line number (low/high)
NI 1470         tay
GP 1480         jsr  acptr
MC 1490         ldx  status ;check status
EH 1500         bne  fini   ;exit on error
MF 1510 ;
JO 1520         jsr  fixflt ;put line number in fpacc#1
IA 1530         ldy  #1     ;print line number
FP 1540         jsr  shwflt ;(which is the block count)
LG 1550         lda  #32    ;print a space
PO 1560         jsr  chrout
IJ 1570 ;
FK 1580 loop    =  *
PI 1590         jsr  acptr  ;get character
IP 1600         php
HC 1610         jsr  chrout ;print it
IB 1620         plp
CE 1630         bne  loop   ;if non-zero then continue
ON 1640 ;
HP 1650         jsr  return ;print carriage return
MA 1660         jsr  stop   ;check stop key
KO 1670         bne  main
GA 1680 ;
KO 1690 fini    =  *
PB 1700         jsr  untlk  ;un-talk
JK 1710         lda  #8
MJ 1720         jsr  listen
OL 1730         lda  #$e0   ;the secondary address
IL 1740         jsr  second ;is 'or'ed with $e0
EK 1750         jsr  unlsn  ;to indicate a file to be closed
FO 1760         rts         ;return to basic
```

Commodore External RAM Expansion Cartridges

Dale A. Castello
Montgomery, AL

Transfer commands for your external storage area!

Editor's Note: *Although the 1700 and 1750 Expansion RAM modules will work on the C64, they draw about 200 milliamps and the C64 power supply can not handle the extra load. Should you wish to use either of these with the C64, you'll need a higher output power supply. However, the Commodore 1764 External RAM Expansion comes with a replacement power supply, and Dale's software will also work with the 1764. Naturally, the C128 supplies ample power for operating the expansion RAM in 64 mode with Dale's program.*

After many months of anticipation, the expansion RAM cartridge for the C128 is finally available at local stores and by mail. It comes in two versions: the 1700 contains 128K bytes of memory and the 1750 contains 512K bytes. Only the 1750 is readily available. This memory expansion cannot be directly addressed like the resident memory banks internal to the C128. Instead, access is established through the I/O space from $DF00 to $DF0A. Because the expansion cards use the computer's direct memory access (DMA) capability, a memory bank containing the C128 I/O space does not need to be turned on during the actual transfer. Commodore recommends that transfers be done with the 1MHz clock rate so as to avoid conflicts with the memory bus access. Transfers at 2MHz can be done, if the VIC screen is blanked and the instruction following the command execution does not make a write to memory.

The card offers four functions:

(1) FETCH – transfers from external RAM to internal RAM
(2) STASH – transfers from internal RAM to external RAM
(3) SWAP – exchanges internal and external RAM
(4) VERIFY – compares internal and external RAM

C128 BASIC implements the first three of these functions. The fourth function may be executed through use of pokes in C128 mode. A program to implement all four of these functions in C64 mode is discussed later in this article.

Physical Layout

The expansion RAM chips and DMA controller are housed in a C128–colored, plastic unit which is 5 1/4 inches wide and extends 4 1/2 inches behind the computer when plugged into the expansion port. There is no edge connector on the unit to permit other bus devices to be plugged into it. Inside the case are the DMA controller chip and 16 memory chips. The chips are either 64K by 1 bit for the 1700 or 256K by 1 bit for the 1750. Wire straps on the card indicate that Commodore designed the circuit card for 128K, 256K, and 512K byte configurations.

Internal Registers And Operation

The external RAM controller appears at I/O addresses $DF00 through $DF0A. Of these eleven addresses in the controller: one is for status, three for control, and the rest for addresses. All of the registers are read/write except the status register which is read only.

In order to activate an operation, the starting memory locations in internal and external RAM, the block size, some special options, and the command must be written to the controller. The actual transfer occurs either immediately following the write of the command or after the next bank switch of the C128. The latter feature permits the C128 banks to be reconfigured prior to the transfer so that memory under I/O may be transferred.

The internal computer RAM starting address is placed in $DF02/$DF03 in normal low/high byte order. The C128 bank configuration must be set in $FF00 or in location 1 if you are using a C64.

The external RAM is banked in increments of 64K bytes. Because it is only possible to address 64K memory locations using two bytes, the starting location in the external RAM requires three locations. The location is given in normal low to high order in $DF04 through $DF06. The values in $DF06 are limited to 0–1 for the 1700 and 0–7 in the 1750. If the block of data to be transferred extends across a bank boundary, the DMA controller automatically increments the bank register.

The size of the transfer is set in locations $DF07 and $DF08 in normal order. Transfers are limited to 64K bytes with all block sizes normal except size value of zero means 64K.

The DMA controller also permits an interrupt to be set when it completes its operation. Because the DMA controller disables normal CPU processing on the C128, this capability is not used

on the C128. This means the interrupt must be processed by the user's program and will not be handled by the operating system. Location $DF09 is the interrupt mask for the controller. It works in the same way as the interrupt mask registers on other I/O devices. During a write, mask bit 7 determines if the interrupt will be enabled or disabled. Two conditions may be set: bit 6 causes an flag at the end of an operation and bit 5 sets a flag if a verify error occurs. The actual interrupt event is signalled by the setting of bit 7 in the status register. A read of $DF00 (the status register) will indicate which event caused the interrupt. Bits 6 and 5 of the status register have the same meaning as in the interrupt mask register. A read of the status register is destructive and will clear bits 5–7.

The status register has one more bit of interest. Bit 4 indicates whether a 1700 or 1750 is attached. If the bit is set, a 1750 is attached; otherwise, a 1700 is attached. The last two registers determine the operation of the controller. The register at $DF01 is called the command register and the one at $DF0A is the address control register.

During normal operation you will want both the internal and external addresses to increment as each byte is transferred. There are special cases where you would want to hold one address constant, such as a direct transfer with I/O. Bits 6 and 7 at $DF0A are normally zero which permits both addresses to increment. If bit 7 is set, the C128 address will be fixed. If bit 6 is set, the external RAM address will be fixed.

The register at $DF01 is the command register. It is set after all the other registers are set and determines the function to be performed. All bits must be set during a single write to the register. Bit 7 must always be set and it executes the function specified by the other bits and registers. Setting bit 5 enables the auto–reload feature. This causes the initial internal memory start address, the external memory start address, and block length to be reset after the function is completed to their values before the function. This option is of value if the same addresses are used repeatedly, such as the VIC screen in computer memory. The user need only set the addresses which change between commands. A disadvantage of the auto–reload feature is that the reload will occur even after an error is found during a verify operation. This destroys the address pointers to the errored byte.

Setting bit 4 enables the bank switch delay. When selected, the actual DMA transfer will not occur until the C128 bank is set by a store to location $FF00. This is the mode of operation used by C128 BASIC. It will not function properly in C64 operation. Finally, bits 0–1 of the command determine the function:

Bit	Function
0 0	Transfer from internal to external RAM (STASH)
0 1	Transfer from external to internal RAM (FETCH)
1 0	Exchange internal and external RAM (SWAP)
1 1	Compare internal and external RAM

After an operation is complete, the address registers will advance by the length register. The length register will be set to one unless auto–reload is enabled. If there is a bad byte detected during a verify operation, the internal and external address registers will point to one location beyond the mismatch.

C64 Operation

There are no commands built into the C64 BASIC to support the external RAM. Therefore, the program accompanying this article provides a BASIC extension of four new commands. The syntax of the commands is the same as in the C128 BASIC except an "@" has been added in front of each. The "@" is part of the keyword and no space should follow it. Any valid expression may be used for the arguments.

@FETCH <length>,<C64 addr>,<RAM addr>,<RAM bank>
@STASH <length>,<C64 addr>,<RAM addr>,<RAM bank>
@SWAP <length>,<C64 addr>,<RAM addr>,<RAM bank>
@COMPARE <length>,<C64 addr>,<RAM addr>,<RAM bank>

Where:
<length> range 0–65535 is size of memory block (0 means 64K)
<C64 addr> range 0–65535 is starting loc. in computer memory
<RAM addr> range 0–65535 is starting loc. in expansion mem.
<RAM bank>is expansion memory bank range 0–1 for 1700
range 0–7 for 1750

The wedge is activated by SYS 52992 and deactivated by SYS 53020. Care has been taken to permit other wedges to coexist with the expansion RAM wedge provided it is the last wedge activated. The code has been compacted so that it fits in $CF00–$CFFF.

Applications

The application program provided in this article will permit the graphics examples contained on the expansion–RAM demonstration disk to be executed on a C64, provided changes are made to C128 tokens and the graphics screen is properly positioned. Other graphics programs may also be modified. The author is currently working on a virtual disk which will permit some graphics adventure games to be played without disk access.

The availability of the space of three single sided disks at 1MHz transfer rates permits a entirely new realm of games and applications to be considered. One application I use is to place my assembler on RAM and "fetch" it into memory when ever I am ready to run it. I have also written a package to copy and modify text adventure games to use the external RAM. Text adventure games which have a lot of disk access come "alive" when RAM instead of disk is used. High speed, single drive copying of filled, single and double–sided disks without disk swapping is great.

Figure 1: C64/C128 Expansion RAM Register Definition

		7	6	5	4	3	2	1	0
$DF00	Status	Interrupt	End Function	Verify Error	512k RAM	Version Number			
$DF01	Command	Execute	Reserved	Auto–Load	No $FF00	Reserved	Reserved	Transfer Type 0–3	
$DF02	C128 Start	Low–Byte							
$DF03	Address	High–Byte							
$DF04	External	Low–Byte							
$DF05	RAM Start	High–Byte							
$DF06	Address	Bank–Byte 0–1(1700) 0–7(1750)							
$DF07	Block	Low–Byte ($0000 means $10000)							
$DF08	Length	High–Byte							
$DF09	Intr.Mask	On/Off	End Function	Verify Error	reserved				
$DF0A	Addr.Cntrl	Fix C128 Add	Fix RAM Addr	reserved					

Expansion RAM Commands: BASIC Loader

```
LN 1000 rem save' '0:xram64.bas' ',8
KF 1010 rem ** this program will create
HL 1020 rem ** a load and run module on
HF 1030 rem ** disk called ' 'xram64.obj' '
OF 1040 open 15,8,15: open 8,8,1,' '0:xram64.obj' '
PJ 1050 input#15,e,e$,b,c: if e then close 15
        : print e;e$;b;c: stop
LD 1060 for j=52992 to 53244: read x: print#8,chr$(x);
        : ch=ch+x: next: close8
GO 1070 if ch<>30308 then print ' 'checksum error!' '
        : stop
KO 1080 print ' '** finished! **' '
-- 1090 print ' 'load xram64.obj,8,1 and sys52992 to enable' '
GJ 1100 print ' 'sys53020: rem to disable' '
GF 1110 end
EN 1120 :
FC 1130 data   0, 207, 162,  70, 160, 207, 204,   9
LI 1140 data   3, 240,  18, 173,   8,   3, 141,  68
JK 1150 data 207, 173,   9,   3, 141,  69, 207, 142
IA 1160 data   8,   3, 140,   9,   3,  96, 174,  68
FE 1170 data 207, 172,  69, 207, 200, 240,   7, 136
LP 1180 data 142,   8,   3, 140,   9,   3,  96,  83
PH 1190 data  84,  65,  83, 200,  70,  69,  84,  67
ID 1200 data 200,  83,  87,  65, 208,  67,  79,  77
AP 1210 data  80,  65,  82, 197,   0,  76, 255, 255
MK 1220 data 160,   0, 132,   2, 200, 177, 122, 201
FG 1230 data  64, 208, 242, 162,   0, 200, 177, 122
JA 1240 data  56, 253,  45, 207, 208,   3, 232, 208
JD 1250 data 244,  56, 233, 128, 208,   2, 240,  17
DE 1260 data 189,  45, 207,  48,   5, 240, 214, 232
IN 1270 data 208, 246, 230,   2, 232, 160,   1, 208
OM 1280 data 220, 200, 152,  24, 101, 122, 133, 122
CE 1290 data 144,   2, 230, 123,  32, 245, 207, 140
EP 1300 data   7, 223, 141,   8, 223,  32, 242, 207
JN 1310 data 140,   2, 223, 141,   3, 223,  32, 242
MN 1320 data 207, 140,   4, 223, 141,   5, 223,  32
OD 1330 data 242, 207, 201,   0, 240,   3,  76,  72
AF 1340 data 178, 173,   0, 223,  41,  16, 240,   4
OI 1350 data 192,   8, 144,   4, 192,   2, 176, 238
EE 1360 data 140,   6, 223, 165,   2, 160,   0, 140
HL 1370 data  10, 223, 140,   9, 223, 120, 162, 245
EF 1380 data 164,   1, 134,   1,  44,   0, 223,   9
EF 1390 data 144, 141,   1, 223, 165, 122, 208,   2
MJ 1400 data 198, 123, 198, 122, 173,   0, 223, 141
AJ 1410 data  12,   3, 173,   2, 223, 141,  13,   3
GP 1420 data 173,   3, 223, 141,  14,   3, 132,   1
JM 1430 data  88,  76,  67, 207,  32, 253, 174,  32
OB 1440 data 158, 173,  76, 247, 183
```

PAL Source Listing

```
EN 1000 rem save' '0:xram64.pal' ',8
GF 1010 rem ** pal 64 format **
LJ 1020 open 8,8,1,' '0:xram64.obj' '
HN 1030 sys700
DD 1040 .opt o8
HD 1050 *=$cf00
KJ 1060 ;
LH 1070 ; a program to implement external
IP 1080 ; ram function on a c-64 or
PB 1090 ; c128 in c64 mode
CM 1100 ;
IP 1110 ; dale a. castello
LJ 1120 ; 5964 oakleigh rd
DB 1130 ; montgomery al 36116
KO 1140 ;
CF 1150 ; implements basic extensions
KC 1160 ; @stash <bytes>,<addr1>,<addr2>,<bank>
HO 1170 ; @fetch <bytes>,<addr1>,<addr2>,<bank>
MN 1180 ; @compare <bytes>,<addr1>,<addr2>,<bank>
CN 1190 ; @swap <bytes>,<addr1>,<addr2>,<bank>
GC 1200 ;
NA 1210 ; where
ND 1220 ; <bytes> = number of bytes to transfer 0-65535
MM 1230 ;                0 => 65536 bytes
JK 1240 ; <addr1> = computer start address 0-65535
JG 1250 ; <addr2> = ram start address 0-65535
MP 1260 ; <bank>  = ram bank number
KN 1270 ;              0-1 for 1700
PO 1280 ;              0-7 for 1750
AI 1290 ;
EE 1300 ;  activate    sys 52992 ($cf00)
FN 1310 ;  deactivate sys 53020 ($cf1c)
OJ 1320 ;
FF 1330 ; on exit
JD 1340 ; areg  status  $20 okay
LB 1350 ;               $40 verify error
GM 1360 ;
CA 1370 ; xreg/yreg last computer address
KN 1380 ;
GG 1390 cmd     =   2       ;expansion command
DE 1400 txtptr  =   $7a     ;current byte of basic text
FD 1410 areg    =   $30c    ;storage of a reg
LM 1420 xreg    =   $30d    ;storage of x reg
NN 1430 yreg    =   $30e    ;storage of y reg
DO 1440 igone   =   $308    ;basic token eval
```

```
PL 1450 exp     =   $df00       ;dma controller
OJ 1460 c64     =   exp+2
DF 1470 ram     =   exp+4
LA 1480 bank    =   exp+6
LK 1490 leng    =   exp+7
CF 1500 ;
JL 1510 active  = *
IP 1520         ldx #<parse
CA 1530         ldy #>parse
LN 1540         cpy igone+1     ;if page $cf
PD 1550         beq inpl        ;already installed
OI 1560 ;
HG 1570         lda igone
KG 1580         sta oldvec+1
HI 1590         lda igone+1
CI 1600         sta oldvec+2
JC 1610         stx igone
DE 1620         sty igone+1
EN 1630 ;
IN 1640 inpl    = *
OF 1650         rts
CP 1660 ;
HL 1670 inact   = *
MO 1680         ldx oldvec+1
OP 1690         ldy oldvec+2
DE 1700         iny             ;if $ff is hi addr
FD 1710         beq nogo        ;don't restore
OC 1720 ;
NH 1730         dey
LK 1740         stx igone
FM 1750         sty igone+1
GF 1760 ;
JK 1770 nogo    = *
AO 1780         rts
EH 1790 ;
HB 1800 table   = *
EH 1810         .asc ''stas''
AD 1820         .byte $c8
GF 1830         .asc ''fetc''
EE 1840         .byte $c8
ML 1850         .asc ''swa''
EF 1860         .byte $d0
HN 1870         .asc ''compar''
BF 1880         .byte $c5,0
IN 1890 ;
LB 1900 oldvec  = *
LE 1910         jmp $ffff       ;address set to old error vector on activation
GP 1920 ;
IN 1930 parse   = *
HJ 1940         ldy #0          ;scan basic text
OF 1950         sty cmd         ;initial command number
ND 1960         iny             ;point to next character
CL 1970         lda (txtptr),y
IB 1980         cmp #''@''
NN 1990         bne oldvec      ;no leading @
GE 2000 ;
CF 2010         ldx #0          ;init table pointer
KF 2020 ;
MO 2030 nxt     = *
GP 2040         iny             ;get next input character
CA 2050         lda (txtptr),y
NI 2060         sec
PA 2070         sbc table,x     ;check text
MK 2080         bne last        ;may be shifted
AK 2090 ;
HL 2100         inx             ;okay so far
PN 2110         bne nxt         ;loop for next match
OL 2120 ;
FL 2130 last    = *
EO 2140         sec             ;check for shifted
IL 2150         sbc #$80        ;check for shifted
JG 2160         bne skip        ;character
AP 2170 ;
KM 2180         beq found       ;matchs string
EA 2190 ;
NC 2200 ; no match found so advance to
HC 2210 ; next command string
CC 2220 ;
JB 2230 skip    = *
JJ 2240         lda table,x
FB 2250         bmi nxcmd       ;reached shifted char
KE 2260 ;
KJ 2270         beq oldvec      ;error end of table
OF 2280 ;
OM 2290         inx
GH 2300         bne skip        ;keep going
MH 2310 ;
CD 2320 nxcmd   = *
PD 2330         inc cmd
AA 2340         inx
CB 2350         ldy #1          ;dim in basic text
JH 2360         bne nxt         ;search next command
IL 2370 ;
NM 2380 ; we have found the match
EN 2390 ; read parameters
GN 2400 ;
DN 2410 found   = *
KD 2420         iny             ;update basic pointer
FD 2430         tya
OP 2440         clc
EC 2450         adc txtptr
KH 2460         sta txtptr
AL 2470         bcc nopage
GC 2480 ;
MD 2490         inc txtptr+1
KD 2500 ;
JK 2510 nopage  = *
KN 2520         jsr getint      ;get # bytes
JD 2530         sty leng
FK 2540         sta leng+1
BI 2550         jsr arg         ;get c64 memory start
BF 2560         sty c64
EN 2570         sta c64+1
KC 2580         jsr arg         ;get external ram start
PN 2590         sty ram
CG 2600         sta ram+1
IB 2610         jsr arg         ;get bank
IO 2620         cmp #0          ;check if out of range
KB 2630         beq limit
GM 2640 ;
ED 2650 toobig  = *
AA 2660         jmp $b248       ;illegal quantity
EO 2670 ;
GN 2680 limit   = *
HK 2690         lda exp
LM 2700         and #$10        ;check ram size
HF 2710         beq r128
GB 2720 ;
BE 2730         cpy #8          ;max bank for 512k +1
NN 2740         bcc inside
ED 2750 ;
HG 2760 r128    = *
KF 2770         cpy #2          ;max bank for 128k +1
IO 2780         bcs toobig
MF 2790 ;
MM 2800 inside  = *
KN 2810         sty bank
JA 2820         lda cmd
JF 2830         ldy #0
OF 2840         sty exp+10      ;inc pointers
OB 2850         sty exp+9       ;no interrupts
FI 2860         sei             ;open ram
HO 2870         ldx #$f5        ;under basic and kernel
NK 2880         ldy 1           ;old value
IN 2890         stx 1           ;temp value
PD 2900         bit exp         ;reset dma controller
DD 2910         ora #$90        ;form command
CL 2920         sta exp+1
JK 2930         lda txtptr      ;dim in basic text
KC 2940         bne notb
MP 2950 ;
BO 2960         dec txtptr+1    ;page boundry
AB 2970 ;
OC 2980 notb    = *
MH 2990         dec txtptr      ;single byte
DL 3000         lda exp         ;return result
ID 3010         sta areg
DH 3020         lda c64         ;return last address
KL 3030         sta xreg        ;accessed in computer
MG 3040         lda c64+1
IH 3050         sta yreg
LP 3060         sty 1           ;restore ram configuration
JP 3070         cli             ;interrupts on
OK 3080         jmp oldvec      ;back to basic
II 3090 ;
HG 3100 ;subroutine to evaluate argument
MJ 3110 ;
OM 3120 arg     = *
HP 3130         jsr $aefd       ;must have comma
KL 3140 ;
DC 3150 getint  = *
FD 3160         jsr $ad9e       ;eval expression
LA 3170         jmp $b7f7       ;fix it
CO 3180 ;
CF 3190 .end
```

In The CP/M Mode

Clifton Karnes
Greensboro, NC

I'd like to discuss several things in this article. First, there are some exciting software packages in the public domain, including an excellent text editor/word processor and a spelling checker. Second, I want to talk about configuring your C128 CP/M keyboard with KEYFIG (transient utility on the CP/M system disk). Believe me, if you're not using KEYFIG, you're working too hard. Third, I want to djscuss some commercial CP/M packages that I think C128 CP/M users will be interested in.

VDE 2.2

This public domain text editor is the latest in the line that began with the famous VDO (for Video Display Oriented) in 1982. Before VDO, most text editors were line oriented. If you've agonized over CP/M's ED you know what a line oriented editor is--it's a pain. (You may have heard the CP/M cry, "Better dead than ED.") VDO changed all that.

The original VDO was developed by Richard Fobes and published in the September and October 1982 issues of Byte. Soon after, variations and improvements began to appear. The latest VDO is James Whorton's VDO 2.5(b), which I used before discovering VDE.

VDE has its roots in VDO but is so much enhanced that its author, Eric Meyer, decided that it was best to change the name slightly and call it VDE. VDE 2.2 was completed August 1986, so is quite recent. It is a 58K library file which contains two versions of VDE: VDE-M, for memory-mapped systems, and VDE, for terminals (the version the C128 uses). In its distributed form VDE is configured for an Osborne, but thanks to the 128's extended terminal emulation, this version works fine.

Why VDE?

What's so good about VDE? First, it's only 9K. This means it doesn't take up much disk space, and it loads quickly. When you're doing program development and going back and forth in the edit/compile/link/run cycle, this fast loading can be a tremendous time saver. VDE is also fast working and full-featured. It has full-screen editing, windowing, horizontal scrolling, automatic pagination (which can be turned off), file directory, block operations, macro functions, find/replace, undelete, several user definable options, and (I've saved the best for last) VDE has word processing capabilities. The word processing may only consist of word wrap and some simple formatting, but it is all many people will ever need, and it's all most people need most of the time.

The three VDE files that are important to C128 users are VDE-22.COM (the text editor), VDE22.DOC (documentation), and VDE22OV.ASM (an overlay file to configure VDE). As I mentioned earlier, VDE works "right out of the box" for C128 CP/M, but you may want to fine-tune it for your preferences. You can do this either by editing the overlay file or, more simply, by using the addresses supplied in the overlay to change the values with SID. For example, if you're using the 80-column RGB signal with a composite monitor you'll want to reverse the high and low intensity values. You may also want to change tabs (they can't be changed from inside the program). The most important configuration you'll have to do is for your printer. You can define three toggles and four switches. These define special codes that will be sent to your printer for things like underlining, boldface, italics, and so on. Macro key definitions can be hardwired in with SID, but there's a much easier way to do it which is explained in the documentation.

Using VDE

VDE can be invoked with or without specifying a filename. VDE gives a status line at the top of the screen which shows filename, page, line, column, and mode (insert or overwrite). Pressing ESC-? will give a menu of commands.

You can set up ten macros with VDE. Each macro can be up to 65 characters long. These macros can be either temporary or they can be saved. The documentation with VDE explains the procedure for saving your macros.

Two of the nicest commands in VDE are the ones that set the right and left margins. These enable wordwrap and give VDE its word processing capabilities. VDE also has automatic top and bottom page formatting (which can be turned off if you like), as well as centering, right margin alignment, and several other text processing commands. There are no provisions for headings or other fancy features, but most writing applications don't call for these, anyway.

VDE reminds me of the motto of the old Dr. Dobbs Journal: "Running Light Without Overbyte." Eric Meyer deserves the thanks of all CP/M users for creating such an excellent software package and putting it in the public domain.

Spell 2.1

Spell is a public domain spelling checker. Version 2.1 was released in 1985, but the Spell program has a history going back to 1982 and before. The current version is by Michael C. Adler, and has its roots in the work of William Ackerman at MIT

I downloaded Spell 2.1 from CompuServe's CPMSIG DL1 as a 124K library file called SPEL20.LBR. The three necessary files to run Spell are SPELL.COM (the spelling program), DICT.DIC (the dictionary), and SPELL.DOC (the documentation). The library also contains the Z80 assembly language source code for SPELL.COM and a program (DICCRE10.COM) to create a compressed dictionary file from a raw file.

Spell not only looks for words in its main dictionary, but in a pre-named user dictionary (SPELL.DIC) and it can be given command-line options to have it search any user-created dictionary.

To use Spell you just specify the file you want to it to check. Spell looks in its main dictionary, user dictionaries, and any specified dictionaries for the words in the file. Words not found are marked. The marking character is a null (↑', ASCII 0), but this can be changed if you don't like it. If you're using WordStar you can use the ↑QL command to correct the marked file painlessly. If you're using another word processor you need to go through the text and erase each marker and change the word if it's spelled incorrectly.

Spelling Bee

To find out how good Spell is I decided to test it against The Word Plus and Perfect Speller. The Word Plus is a collection of correction and writing aids, of which the spelling checker is one part. It is considered by many to be the best program available. Perfect Speller comes with Perfect Writer.

There are two ways a spelling program can go wrong: it can fail to catch a misspelled word or it can mark a word as misspelled that isn't. Marking words that aren't misspelled as wrong can be corrected by adding those words to the user dictionary. Missing words that are not spelled correctly is the result of data structures for the dictionary and algorithms used and it's performance can't be improved by the user. It is important to consider both of these errors. The best spelling checkers will give balanced performance in both areas. Consider, for example, a spelling checker that marks every word as misspelled. It would get 100% in catching errors, but would fare poorly in the other area. A checker that marked no words would be 100% accurate in not marking correctly spelled words as wrong, but it would get a zero in catching errors. For those interested in the design and history of spelling checkers there is an interesting chapter in Jon Bently's Programming Pearls "A Spelling Checker" (New York: Addison-Wesley, 1982), pp. 139–150.

To test the checkers, I used a list of 75 frequently misspelled words along with their correct spellings, making a total of 150 words. The list comes from a test (intended for human spellers) in Harry Shaw's Spell it Right, collected in Read, Write and Spell it Right (New York: Greenwich House, 1982), pp. 476–478. I was surprised by the results:

Speller	Misspelled Words Not Caught		Correctly Spelled Words Marked as Incorrect	
	Number	Percentage	Number	Percentage
The Word Plus	2 (1)	97% (99%)	0	100%
Perfect Speller	24	68%	5	93%
Spell	2 (1)	97% (99%)	9	88%
Paperback Speller	1	99%	40	47%
Easyspell	1	99%	13	83%

As you can see, The Word Plus gave the best all round performance. And to be fair, one of the two words it let slip is given in the dictionary as acceptable. What is really impressive is its 100% accuracy in the second column. This is a testament to the thoroughness with which this product was designed. Perfect Speller is a surprise. It must be judged a failure. Nothing is going to improve the number of misspelled words it catches. The next surprise is Spell. It did as well as The Word Plus in the first column. (One of its "errors" was also deemed as acceptable by the dictionary.) True, it didn't do as well in the second column as The Word Plus, but some work on a user dictionary will ease that problem.

To provide a little more perspective I ran the spelling test with two other popular spellers for the 64/128 side of the machine. I'll let you draw your own conclusions on these results.

I am delighted with Spell and I think it is the perfect complement to VDE. These two packages do so much—for free (almost).

Make Your Keyboard Sing With KEYFIG

KEYFIG is one of the nicest things about the C128's CP/M. And one of the best uses of it is to reconfigure your numeric keypad as cursor control and editing keys. With KEYFIG you can define a separate diamond-shaped area for cursor movement, and use the nearby keys for common editing functions. The diamond-shaped cursor control layout has been judged by most to be the best ergonomic design.

First, some background. The keyboard definitions are saved as part of the CP/M+.SYS file, and, when it is loaded, the definitions are stored in Bank 0. This means that you can have several different logical keyboards—as many as you need. For example, if you use several text editors and word processors, all of which used different cursor and editing commands, with KEYFIG, you can create different keyboard definitions for each application. The cursor-up command may be Control-F in one application, Control-E in another, and Control-W in another. If you configure your numeric keypad for use as cursor control keys, you can have keypad-8 be cursor up no matter what the code for the application might be. This makes life so much simpler.

To get started, type KEYFIG at the "A>" prompt. You'll be asked if you want help. You don't now, but you may want to review some of these topics later. Press "n" and you'll be asked which definitions you want to use. Since you haven't created any definitions yet, you'll use the default definitions. Move the cursor to that selection and press Return. (While in KEYFIG you'll need to use the 128 arrow keys to scroll through your choices.) Now you're given a choice of three things to do: edit a key, assign colors, or exit and save our workfile. To start, you want to edit keys, so make this selection. To begin, let's define the keypad's 8, 4, 6, and 2 as cursor up, left, right, and down. To edit a key, just press it, so press "8". You'll see that the key has several values: its normal value, a shifted value, a control value, and a Commodore-key value. You want to alter the normal (top) value first, so make this selection. You're presented with another menu offering various types of assignments. You can assign a new character, a string, a color, a special function, or a hex value. For this key you'll be assigning a single control character, so make that selection.

Now, press the control code for the assignment you want to make—Control-E for WordStar-like editors. That key has been reconfigured. You can define the rest of the the cursor keys just like "8" by supplying the appropriate control values for each key.

Defining strings is just as easy. After making the choice for string assignment, you'll be presented with a list of the 32 available strings. Some of these will already have been assigned. Scroll through and make a selection and press Return. Then simply enter the string. To end strings with a carriage return use Control-M.

What follows is one way to configure your keypad by function:

Normal Value		Shifted Value		Control Value	
Key	Function	Key	Function	Key	Function
(9)	page up	(9)		(9)	
(8)	cursor up	(8)	3 lines up	(8)	
(7)	home	(7)	top of block	(7)	top of file
(6)	cursor right	(6)	word right	(6)	
(5)	end of line	(5)	beginning of line	(5)	
(4)	cursor left	(4)	word left	(4)	
(3)	page down	(3)		(3)	
(2)	cursor down	(2)	3 lines down	(2)	
(1)	bottom screen	(1)	bottom of block	(1)	bottom of file
(0)	insert line	(0)		(0)	
(.)	mark block	(.)	mark block end	(.)	
(+)	insert/overwrite	(+)		(+)	
(–)	delete char	(–)	delete to end of line	(–)	delete entire line

Keys with nothing beside them are not defined. The normal values comprise all the heavily used cursor movement and editing commands. The shifted values (when they exist) are intensifications of the normal values, and the control values are further intensifications. This is only a guide. Your word processor or other application may not have all these functions (the one I'm writing this with doesn't), or it may have more.

Pressing the Commodore key (C=), which acts as a CAPS toggle in C128 CP/M, will give you the numeric values for the keypad. The above configuration will create an easy-to-use, diamond-shaped cursor control station. In addition to the keypad configuration I have found the following redefinitions helpful:

(Control–HELP) (Special function) BOOT – This gives you an easy way to reboot CP/M without resetting, and when you want to go from CP/M to 128 mode you can press Control–HELP instead of resetting.

(ALT) (String) B:↑M – on a two drive system this will save two keystrokes when you want to change to drive B:

(Control–ALT) (String) A:↑M – to change back to A:

(F1) (String) SD↑M – I use SD.COM (SuperDirectory) most of the time instead of DIR. This makes it just a keystroke away.

When you've finished configuring your keyboard you need to save the new definitions. You can save them as the current definitions — to try them out to see if you like them — or save them on the boot disk. If you save them as the current definitions, you'll need to enter KEYFIG again later and save the current definitions permanently to your boot disk.

CP/M Software Update

I'd like to briefly discuss some CP/M software, both old and new, that I think is important to C128 CP/Mer's. Details on the packages discussed are given at the end of this article. First, I want to mention two pieces of hardware: the 1700 and 1750 expansion modules. These memory cards work as RAM disks in CP/M and are great. Anything on the RAM disk is accessible immediately, just like a resident CP/M command. Going through the edit/compile/link/run cycle with one of these is so pleasant. It's almost as easy as working with an interpreter. And it's so quiet. One 128K RAM disk and one 1571 would make an excellent CP/M system for about $100 less than a two–drive system would cost.

There are two new releases by Commodore CP/M Engineering made available on CompuServe recently. One is the source code for the December 6 BIOS. (The BIOS source that comes with the DRI offer is an earlier version.) Also recently made available is a new version of FORMAT. The new version is smaller, can format disks on either drive (the original only formatted disks on drive A:), and (most important) it can format disks in any of the formats the C128 can read. All the formats haven't been verified, but the KAYPRO and CPM–86 formats work fine. The others should, too.

Perfect Writer

As for commercial CP/M software, I've tried several packages recently that work well on the C128 and one that was a disappointment. Perfect Writer is a word processor that has been released by Commodore for the 128 in CP/M mode. (It used to be bundled with Kaypros, but WordStar is now.) The Perfect Writer package comes with Perfect Speller (which you've already met) and Perfect Thesaurus for the low price (on the street) of around $50.

First, Perfect Writer is what is called an EMACS–style editor as opposed to a WordStar–style editor. These distinctions refer to the editor's command structure. Perfect Writer 2.0 not only has EMACS–style commands but pop–up menus. These menus can be turned off if you like and just the commands used. I don't want to give a complete review of Perfect Writer here, but there are several things a potential buyer should know. It's huge: slow loading and slow working. It uses a disk swap file. Every so often everything stops while text in memory is saved to the swap file on your disk. There is a quick print function, but printing documents that use any advanced formatting functions is a different program (on a different disk) that must be loaded and run.

One annoying thing is that Perfect Writer doesn't use console input, so strings assigned with KEYFIG won't work. This means you can't do much in the way of reconfiguring your keypad to make Perfect Writer easy to use except implement the single cursor movement keys and the other single–keystroke commands. Perfect Writer doesn't automatically reformat after inserting, either. And Perfect Writer committed what Byte columnist Jerry Pournelle calls, "the one unforgivable sin": it lost text. This happened to me several times. I reported it to Commodore CP/M Engineering but haven't received a reply. It could have been a glitch in my copy, but this seems unlikely. On the plus side, Perfect Writer comes with a superb manual and is loaded with features. And I have talked to some people who love the menus. The menus will make Perfect Writer easy to learn, but with its command structure, and not letting strings from KEYFIG through, it will never be fluid, fast, or flexible.

WRITE, Some Writing Aids, an Assembler, and More

WRITE (Writer's Really Incredible Text Editor, $99.95 from Workman & Associates) is an excellent word processor for the C128 and like a breath of fresh air after Perfect Writer. It is memory–resident and thus has no swap file to slow you down. It's fast, powerful, and a

joy to use. Printer configuration may be a problem, though, so check with Workman & Associates before you buy. Also, install the Televideo 912 terminal instead of Lear-Siegler ADM31. WRITE's previewer works better with this emulation on the C128.

I've already mentioned The Word Plus ($150.00 from Oasis Systems). It is a spelling checker, correction manager, homonym checker, hyphen maker, a huge dictionary, and more. An excellent product. (I understand that The Word Plus is being shipped with NewWord (a word processor like WordStar but better according to most) all for $125. Like getting a discount on The Word Plus and a word processor free.) If all you need is a speller though, consider Spell. Also from Oasis is Punctuation + Style ($125.00). There are two parts to this program. The first is called CLEANUP. It finds extra spaces, mixed capitals--like THis--and such like. PHRASE is the second program and it marks hackneyed expressions and unnecessary words. For example, if you write: "all of the apples looked good," PHRASE would say that "of the" is unnecessary and should be cut. I was amazed at how good PHRASE is. If you do serious writing, it will more than pay for itself.

Write-Hand-Man ($49.95 from Poor Person Software) is a Sidekick-like utility. It lurks in the C128's high memory waiting for you to call it. When you do, the present program is interrupted and WHM offers you a notepad, a calendar, a phonebook, a terminal program, hexadecimal and decimal calculators, and ASCII chart, and more. This is a useful package.

Z80ASM and SLRNK ($49.95 each from SLR Systems) are a Z80 macro relocatable assembler and linker. Z80ASM is fully compatible with Microsoft's M80. Many CP/M assembly language programs were written and are being written with M80 so this is an important feature. Probably the best thing about Z80ASM is that it can produce executable .COM files in one step and it is FAST. And I mean warp-factor five FAST! Z80ASM takes a lot of the bite out of assembly language programming.

Z80DIS is a public domain Z80 disassembler with some interesting features. It can generate its own breakpoints--or at least will try. This means it will decide which parts of the machine code are instructions and which are data. Impressive. The source code for Z80DIS (which the author hasn't released) is over 5000 lines of Turbo Pascal.

I hope to have more detailed reports on some of these products in the future, as well as several that are still in my review queue. Queued items include two packages from Kamasoft: Out-Think and KAMAS. Out-Think is an outline processor and text editor, and KAMAS is an outline processor and programming language. Also, MTBASIC, a multitasking BASIC interpreter/compiler; Fancy Font, by all accounts, THE printer enhancement package; and ASM from MIX Software, which lets you run assembly language subroutines with their C compiler. I have just started working with these packages so I don't have much to report yet. More later.

How to Get the Public Domain Software

All the public domain programs mentioned in this article — VDE, Spell, Format, the December 6 BIOS, and Z80DIS — are available on CompuServe (see below for details), or you may be able to find them on a bulletin board near you, get them from a friend, or from local user's group. If you can't find them, send me an SASE and a formatted C128 single- or double-sided disk plus $3 for each library you want, and I'll copy them for you. There are four libraries: VDE22.LBR, C6DEC.LBR, Z80DIS21.LBR, and SPEL20.LBR (FORMAT is small and will be on every disk).

A note on .LBR and squeezed files. Putting files in libraries and squeezing is a way of making the files smaller, so they'll transfer more quickly, and more unified — one library file instead of several single files. If you download any of these libraries yourself you'll need a de-library utility and an unsqueeze utility, like DELBR and USQ, or NULU, so download these programs, too. If you get the files from me I'll put these utilities on the disk.

Clifton Karnes
2519 Overbrook Dr.
Greensboro, NC 27408

Software Sources Mentioned:

CompuServe CPMSIG

VDE22.LBR	(54K)	(DL1)	public domain
SPEL20.LBR	(124K)	(DL1)	public domain
C6DEC.LBR	(130K)	(DL3)	public domain
Z80DIS21.LBR	(148K)	(DL2)	public domain
FORMAT.BIN	(2K)	(DL3)	public domain

WRITE $99.95
Workman & Associates
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

The Word Plus $150.00
Punctuation + Style $125.00
Oasis Systems/FTL Games
6160 Lusk Blvd. Suite C206
San Diego, CA 92121
(619) 453-5711

Z80ASM $49.95
SLRNK $49.95
SLR Systems
1622 N. Main St.
Butler, PA 16001
(412) 282-0864

Write-Hand-Man $49.95
Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
(415) 493-3735

NewWord $125.00
(includes The Word Plus)
NewStar Software
1601 Oak Park Blvd.
Pleasant Hill, CA 94523
(415) 932-2526

Please Note: Ellis Computing, the source for Nevada BASIC, FORTRAN, COBOL, Edit, etc. has moved. Their new address is:

5655 Riggins Court, Suite 10
Reno, Nevada 89502
(702) 827-3030

Using CP/M Plus User Areas

Adam Herst
Toronto, Ontario
(C)1987 Adam Herst

Making the most of practically nothing.

The number of files that the C–128 can store on a disk is limited by two factors: the storage space available and the maximum number of directory entries. When disk space is limited, the number of files on a disk rarely approaches the maximum and directories remain a manageable size. With the increased storage space available on large capacity disk drives, however, directory listings grow proportionately. The number of files often reaches the maximum allowed and a given file can be difficult to find. CP/M Plus provides an aid to file organization in the form of 'USER AREAS'.

The C–128 can currently use two Commodore disk drives: the 1541 and the 1571. Both can be used in the 128's CP/M mode although the 1541 is far too slow to be of practical use. In CP/M mode, the 1541 can store a maximum of 134 Kbytes in a maximum of 64 directory entries. The 1571 can store a maximum of 336 Kbytes in a maximum of 128 directory entries on a disk in C–128 double–sided format. (In some MFM formats the 1571 can hold more data – the Kaypro IV format can hold 390 Kbytes with a maximum of 128 directory entries – but the benefits of increases in storage space are offset by decreases in disk access speed).

Unmanageable directories are rarely a problem with the limited storage capacity on the 1541. The storage capacity of the 1571 however – more than double that of the 1541, can make for very long directory listings. Finding a particular file, even with CP/M's sorted directories, can be a chore.

CP/M's file naming conventions can also make directories difficult to read. File names in CP/M mode are limited to a maximum of 8 characters compared to Commodore DOS's limit of 16 characters. Trying to create distinct but meaningful file names can become a poetic exercise. While a filename can be modified by a 3 character filetype, you do not always have the option on the filetype that can be assigned. Many filetypes are reserved to designate categories of files and should not be used indiscriminately. The filetype .com, for example, must modify the filename of every executable file, while .sub is reserved for files used by submit.com

The problem gets worse if you are using the 1750 RAM expansion cartridge. This cartridge adds 512K bytes of extra memory to the C–128. Unfortunately, the CP/M operating system is limited to an environment of 64K total memory. To make use of the extra memory, it is configured by CP/M on the 128 to be recognized as a RAM disk with the drive designation M:. This allows for very fast 'disk' access since the 'disk' is really memory. It also provides a 'disk' that is larger than any of the real disks currently available for the 128.

The RAM disk can store a maximum of 508 Kbytes in a maximum of 127 directory entries. Paradoxically, while the storage space is over 50% greater than on the 1571, the maximum number of directory entries is one less than on the 1571.

(The 1750 RAM expansion is indispensable when using CP/M on the C–128. CP/M is a disk–based operating system. It is loaded into the computer when a CP/M session is started and practically all operating system commands must be loaded into memory by CP/M before they can be executed. While CP/M cannot be booted from the RAM disk, the CP/M environment can be customized so that all subsequent commands can be loaded from RAM disk, rather than disk drive, with substantial increases in performance. If you are using CP/M but do not have a 1750 expansion unit, you should seriously consider buying one.

For those with a curious nature – there is a secret hidden within the RAM disk. By now 128 users are aware of the SYS in 128 mode that displays the names of the creators of the 128 and their feelings about the arms race. Von Ertwine, the programmer responsible for porting CP/M to the 128, has also hidden his name in the RAM disk in CP/M mode. To display it, issue the 'show m:[label]' command.)

The RAM expansion cartridge provides the 'disk' with the greatest amount of storage space that is currently available for use with CP/M. That situation will soon change. A number of hard–disks with 10 and 20 megabyte storage capacities were shown at the World of Commodore in Toronto this year. While none of them worked in CP/M mode on the 128, representatives for all of the manufacturers promised that capability in the near future.

More realistic, both in terms of price and availability, is the 1581, a 3.5 inch drive from Commodore. Commodore Canada has confirmed the 1581 will operate in CP/M mode, although CP/M cannot be booted from it. This drive will provide in the neighbourhood of 800K bytes of storage space. No information is available on the maximum number of directory entries. Commodore will be limiting its potential if they do not make provisions for the storage of a greater number of files than on the 1571.

One requirement of storage devices with ever increasing capacities is a directory system that can expand beyond a single, simple sequential list. One popular file management system is the arrangement of files into nested subdirectories starting from a root directory. This arrangement allows for manageable directory listings as well as the extension of the limit on the number of files that can be stored on a disk. While the number of directory entries will still be limited, the number of directories will have increased.

This file management method is used by practically all 'modern' operating systems -- by all, that is, but CP/M Plus. To preserve compatibility with its earlier incarnations, a different file management system is used. Instead of a root and nested subdirectories, CP/M Plus has the ability to divide a disk directory into fixed directories, called user areas. A user area is purely a logical construct – there is no physical user area on a disk. It is only a designation given by CP/M to a file.

Early versions of CP/M – up to version 1.4 – did not provide user area capabilities. User areas were first introduced in CP/M 2.2 in an attempt to solve two problems. The availability of large capacity mass storage devices introduced the directory and filenaming problems discussed above. Secondly, system security and data integrity were becoming problems in multi–user environments.

These problems could have been solved using a system of nested subdirectories. This, however, would have made subsequent versions of CP/M incompatible with their predecessors. None of the then available software was designed to operate in a multi–directory environment. Programs looked for overlays and data files in the current directory since no other directory existed. An operating system that made these programs obsolete would have alienated both program developers and users.

Instead CP/M creates 16 fixed, distinct user areas numbered 0 through 15. A user area can be thought of as a work environment. Most programs and operating system commands are executed within the current user area. Files on a disk are designated as belonging to the user area in which they were created. Access to user areas, and the files associated with them, can be restricted with passwords.

Normally, the files in a user area other than the current one are invisible to the operating system. This allows programs from the pre–user area era to operate without modification. These programs consider the current user area to be the only possible environment and do not attempt to access other user areas. Their drive/file specification syntax does not make provisions for accepting a user area specifier. For these programs to work, all of the supporting program files and data files must be located in the user area from which they were invoked.

Some recent CP/M programs will support operations across user areas. Unfortunately, the majority of the transient commands provided with CP/M Plus do not fall into this category. They do not recognize non–current user areas and will not access files outside of the current one. Their command syntax will not accept a user area specification. Only four CP/M Plus transient commands provide options for specifying user areas: USER, SHOW, DIR and PIP.

The current user area is indicated in the system prompt. A number corresponding to the user area number precedes the current–drive letter in all but user area 0. User area 0 is the default user area and is the current environment when CP/M boots up. In user area 0, no number precedes the current– drive letter. It is not possible to name user areas and reference them as such. All references to user areas through the system prompt must be by number.

User areas can be changed by entering:

du: <cr>

at the prompt, where d is the drive–letter, and u is the user area. The current user area can also be changed using the built–in USER command. Enter:

USER u <cr>

at the prompt, where u is the user area or:

USER <cr>

at the prompt to execute the interactive version of USER.

The numbers of the active user areas (user areas that 'own' files on the currently active disks) and the number of files within those user areas can be determined with the SHOW command. One of the options available with SHOW is USER. (Executing SHOW without any options displays the number of Kbytes left on the disk.) To determine the active user areas enter:

SHOW d:[USER] <cr>

where d is the drive to be examined. The word USER must be enclosed in square brackets as with all CP/M Plus options. In addition, since SHOW is a transient command, it must be located on the current drive in the current user area.

CP/M will respond with output resembling:

```
A: Active User :   1
A: Active Files :  0  1   2   4
A: # of files   : 16  5  12   7

A: Number of time/date directory entries: 32
A: Number of free directory entries:      53
```

The A: followed by the colon indicates that this information is for the disk in drive A. The 'Active User' is the user area number from which the SHOW command was issued, in this case user area 1. 'Active Files' are the numbers of the user areas on the disk in drive A that have files associated with them. The '# of files' are the number of files associated with the user area number directly above them.

Having determined the active user areas, their directories can be displayed in two ways. Issuing the DIR command will display the directory of the current user area. To display the directory of another active user area, you can make it the current user area with the USER command, then issue a simple DIR command. Alternatively, the directories of any active user area can be displayed from the current user area, using one of the options provided with the transient version of the DIR command. Because the transient version of DIR is being used, dir.com must be located on the current drive in the current user area.

DIR will accept a variety of syntaxes to display the directories of various user areas:

```
DIR d:filename[Gu]
DIR d:filename[USER=u]
DIR d:filename[USER=(u,u,. . .)]
DIR d:filename[USER=ALL]
```

where d is the optional drive letter, filename is an optional filename and u is the user number. All of these forms will display directories of user areas other than the current one. The last form of the command will display the directories of all active user areas.

While it is nice to know what files are stored where on a disk, the knowledge is wasted if those files cannot be accessed from the current user area. Fortunately the PIP command can be used to copy files from one user area to another. PIP is a transient command and must be located on the current drive in the current user area whether it is issued with or without options. In fact, issuing the PIP command without specifiers or options will load PIP from disk and execute the program in interactive mode.

The option that lets PIP copy files across user areas is unique among options for CP/M commands. For all other commands, and even for all other options available within PIP, options must be enclosed in square brackets following the source–file specification. With PIP, the user area option alone can be enclosed in square brackets following the destination file specification. This allows files to be copied from a non–current user area to another non–current user area, instead of exclusively to the current user area.

The syntax for the PIP command with this option is:

```
PIP d:destinationfile[Gu] = d:sourcefile[Gu]
```

where d is an optional drive specification and u is the user number.

In addition to being a powerful disk–to–disk file copying program, PIP can also copy files from disk to other devices connected to the system. This always includes a screen and may include a printer if you are fortunate. Using PIP with options to copy files from non–current user areas to the screen or printer can simulate the use of simple commands, such as TYPE, issued from the current user area.

Similar to the way a disk drive is specified in PIP by a letter followed by a colon, a device other than a disk drive can be specified by three letters followed by a colon. The screen is specified with CON: (short for console device) and the printer is specified with LST: (short for list device). These device specifications can be used in place of the d: drive specification in the PIP command syntax shown above. The filename and user area number should be omitted since they have no meaning for these devices.

This is the extent of the CP/M Plus commands that will work with or across user areas. For all other CP/M commands and most commercial programs, the current user area is ignorant of other user areas. With one exception, programs executed in one user area cannot access files in another. Programs located in a non–current user area, again with one exception, cannot be executed from another user area. If you think this sounds limiting, you're right. Compared to a system of nested subdirectories with access to files along directory paths, CP/M Plus's user areas are constraining and short–sighted in their vision of real needs.

The only compromise to utility is an indirect benefit of a characteristic of CP/M versions prior to CP/M Plus. CP/M Plus imposes a distinction between SYSTEM files and DIRECTORY files. DIRECTORY files are visible when a simple DIR command is issued, while SYSTEM files are not displayed. (SYSTEM files can be displayed using the DIRSYS command.) In this way, files that are used only by the CP/M operating system can be hidden, reducing the number of files displayed in a directory listing.

The distinction between SYS and DIR files is both artificial and arbitrary. Files that are accessed by the user through the command line can also be designated as SYS files using the SET command. In addition, files that are used by the system can be designated as DIR files with the SET command. The syntax for SET is:

```
SET d:filename[SYSTEM]
SET d:filename[DIRECTORY]
```

A file in any user area can be designated as a SYS file. The result is to hide those files in directory listings. Files in user area 0, however, take on a special characteristic when they are designated as SYS files. Files in user area 0 can be accessed from any user area when they are designated as SYS files. Executable files with the SYS designation can be executed from any user area. Data files can also be accessed, but only for read operations.

This feature allows user areas to function as simple, single level subdirectories with user area 0 as the root directory. Keeping utilities and application programs in user area 0 and giving them the SYS designation allows data files to be distributed among user areas in a rational, project–oriented manner while avoiding the necessity of keeping copies of each program in every user area.

The file management system in CP/M Plus is primitive when compared to many of the operating systems found on todays high–end computers. User areas do not compare favourably with a system of nested subdirectories with the ability to find a file along a specified directory path. Still, on the C–128, they are a vast improvement over the sequential system used in 128 mode. With a little forethought in file arrangement, user areas can be useful for directory management of large capacity disk drives.

Finally, the usefulness and power of the file management system in CP/M can be extended with the use of a variety of add–on or replacement programs. Two of the most promising that I have come across are ConIX, from Computer Helper Industries Inc., and TurboDOS, from Software 2000.

ConIX, a version of which is available in Shareware, is a replacement CCP that sits on top of the standard CP/M BIOS and BDOS. As its name suggests, it emulates many of the file and environment management functions of UNIX. These include variables, named directories and shell scripts with flow control. Compatibility with existing CP/M programs is high since the CP/M environment remains virtually the same for program execution.

In contrast, TurboDOS 2000 is a replacement operating system for the Z80 chip. The manufacturers claim that TurboDOS corrects many of the flaws of CP/M while maintaining compatibility with CP/M programs. Improvements include named directories with directory path searching. As a replacement operating system, TurboDOS requires considerable effort to install. There is a good chance that it may not work on the C–128 with its unique, schizophrenic architecture.

I have requested evaluation copies of the programs from both manufacturers and will keep you posted on my progress. With a little work the boundaries of the Z80 chip on the C–128 can be stretched to their limits.

Assembly Language Disk Error Recovery

Robert V. Davis
Salina, Kansas

Two Small Disk Utility Subroutines

Here are two small disk utility programs which are actually useless by themselves, but they can make life a bit easier for you if you add one or both to your own machine language routines. They solve a problem which occurs when you either forget to turn on your disk drive or you neglect to insert a disk or even if you remember both those items and forget to close the drive door. Since they use standard kernal calls, they should work with equanimity on any 8-bit Commodore computer.

The first program answers the timeless question, "Is the Disk Drive There?" Although I wrote my version to start in normal BASIC workspace, it could exist in the cassette buffer or in any out of the way RAM space. Simply change the assembly address in line 130 and remove line 140. The program opens the standard 15,8,15 file and initializes the drive. If the disk drive does not respond, the program prints "CHECK DRIVE 8" at the current cursor location. If you change the label DRVNR to 9, it works for that drive as well.

If the drive is plugged in and responds, the program exits without a word. Change the JMP $E37B to go to your own program, perhaps to the next error checking program.

The DISK ERROR CHANNEL program is not new, but it is in SYMASS assembler format, and it too avoids contact with the user unless something has gone wrong. If it finds that the error number is 00, it exits gracefully, without writing a note to the operator. If an error is indicated, it will communicate with the person who neglected the care and feeding of the disk drive.

This program occupies a few bytes in the cassette buffer, and as written, is accessed with a SYS 828.

Although I have not experienced a problem, I took care to retrieve all the disk status message from DOS even if it is not printed to the screen. Some authors have reported DOS confusion if only the error number is read.

Of course it is appropriate to call the program when you attempt open a read file, just to make sure the proper disk is in the drive.

Like the first program, the ERROR CHANNEL routine ends in a JMP $E37B, to BASIC warm start. You should insert your own jump or RTS to the appropriate place in your program.

The usual caution applies concerning the opening and closing of the file for the disk error channel. You should only open it before opening all other files in your program and close it after all other files are closed so as to avoid losing information stored in buffers but not yet written to the magnetic surface. In other words, only do that part of the first program in lines 160 through 260 once in your program and only close the error channel when you are done with disk access. Don't be opening and closing the error channel every time you want to check the disk status.

With the combination of the two programs shown here, you can handle many of the common errors involving the disk drive before they crash your program.

When using SYMASS to assemble into BASIC space, starting at $0801, just load and run SYMASS. . . then type

```
POKE 44,64:POKE 16384,0:NEW
```

and press RETURN. This will set the start of BASIC at the 16K mark and will leave about 14K for your object code.

```
FD  100 sys 700
FG  110 ;    "is the disk drive there?"
MD  120 ; using the symass 3.13 assembler
EH  130 * = $0801
AD  140 .byte $0a,$08,$00,$00,$9e,$32
FA  150 .byte $30,$36,$31,$00,$00,$00
GB  160 ;
CM  170         lda  #15      ;file number
HN  180         ldx  drvnr    ;device number
BG  190         ldy  #15      ;secondary address
GM  200         jsr  $ffba    ;set parameters
 IE 210 ;
AP  220         lda  #3       ;length of filename
JN  230         ldx  #<initfn ;low byte
PO  240         ldy  #>initfn ;high byte
BJ  250         jsr  $ffbd    ;set filename
CL  260         jsr  $ffc0    ;open file
JF  270         bcc  fini     ;if carry clear
CD  280 ;                      everything was ok
AJ  290         ldy  #11      ;eleven characters
FG  300 print1=               *
HN  310         lda  errtxt,y ;get one char
 JI 320         jsr  $ffd2    ;send it to screen
FA  330         dey
LD  340         bpl  print1   ;loop until done
KD  350         lda  drvnr
BH  360         ora  #$30     ;convert to ascii
JB  370         jsr  $ffd2    ;output to screen
CP  380 ;
GN  390 fini    =    *
LG  400         ldx  #15      ;file number 15
```

. . .continued on page 52

Upgrading the Amiga 1000 to 32 bits

Dan Schein
West Lawn, PA

. . .the MC68010 at equal clock frequencies will run from 8% to 50% faster than an MC68000 without any user code changes. . .

In recent months there has been a lot of interest in replacing the Amiga's MC68000 microprocessor with a MC68010. This article will explain to you the advantages of upgrading to a MC68010 and also show exactly how the conversion is performed.

MC68010's come in several types; the suggested types for this conversion are the MC68010L8 or MC68010L10. These two types are the least expensive and easiest to obtain. MC68010's can normally be purchased from an electronics supply house, computer flea markets or shows, and many mail order firms. Usual cost for either of these chips is under $20 (US).

Pin for pin, the MC68010 is compatible with the MC68000 currently in your Amiga. The advantages of the MC68010 are many. Here is a partial listing taken directly from "Motorola's MC68010 Micro Minutes MM–444–002":

"The MC68010 at equal clock frequencies will run from 8% to 50% faster than an MC68000 without any user code changes. The new MC68010 multiply is 14 clocks faster, and the divide is 32 clocks faster than the MC68000. Programs utilizing (or with the potential of utilizing) such operations can obtain an increase in performance easily exceeding 10%. The bottom line is, by upgrading an MC68000 system to an MC68010 system, an increase in system performance is obtained which is equal to that which a system redesign from 10 MHz to 12.5 MHz would provide, but with significantly less design cost and effort." Please note that the Amiga runs at 7.14 MHz and not 10 MHz, so this upgrade would make your Amiga equal to 8.925 MHz.

The catch to all these advantages (come on, now, you knew there had to be one somewhere) is a minor software incompatibility. The MC68000's "MOVE SR,ea" instruction has been made into a privileged operation in the MC68010. What this means is that programs using the instruction "MOVE SR,ea" will cause you to receive a software error, followed by a visit from the Amiga Guru.

The fix for this incompatibility is a fantastic piece of code written by Scott Turner. This software solution stops the Guru from visiting when the "MOVE SR,ea" instruction is used. This software fix is actually a "wedge" that catches privileged instruction violations. The wedge then examines the instruction for "MOVE SR,ea"; if found the wedge replaces that instruction with "MOVE CCR,ea" and resumes execution of the program. This wedge is called "DeciGEL" and is available in the Public Domain usually as an ARC type of file. The ARC file usually consists of the assembled code (only 168 bytes), the assembly language source code, and a short program to ensure that DeciGEL is assembled and linked correctly. This ARC file is available from many sources. The most common sources include Amicus Disk #9, Fish Disk #18, and most of the major commercial database services. Local BBS services may also have DeciGEL available for downloading; one such BBS is PhilAMIGA (215–533–3191) where all three DeciGEL files are available for downloading in the form of one ARC type file called "DeciGEL.arc".

Use of DeciGEL could not be easier. Simply place DeciGEL into the root directory of your Workbench disk and edit your Startup–Sequence (found in the S directory) to include a call to DeciGEL. (Do not change it just yet, though). The following is an example:

```
echo "Workbench disk. Release 1.2"
echo "  "
echo "Use Preferences tool to set date"
echo "  "
DeciGEL
LoadWB
endcli > nil:
```

The Startup–Sequence can easily be modified using Ed. Ed is a text editor supplied on your Amiga Workbench disk. For instructions on using Ed consult the Amiga DOS Users Manual, or Volume 6, Issue 6 of Transactor.

Now it's time for the actual conversion of your Amiga. The tools you will need are a Phillips screwdriver, a small straight screwdriver, and a chip puller. But first, a few words of caution. The circuit board and its components inside the Amiga are very fragile and very, very sensitive to static. Opening your Amiga to perform this upgrade will void your warranty, so wait till it's over (90 days is not that long). Caution and common sense are all you should need; take your time and be careful. If you want to be safe, have an experienced technician perform the upgrade for you.

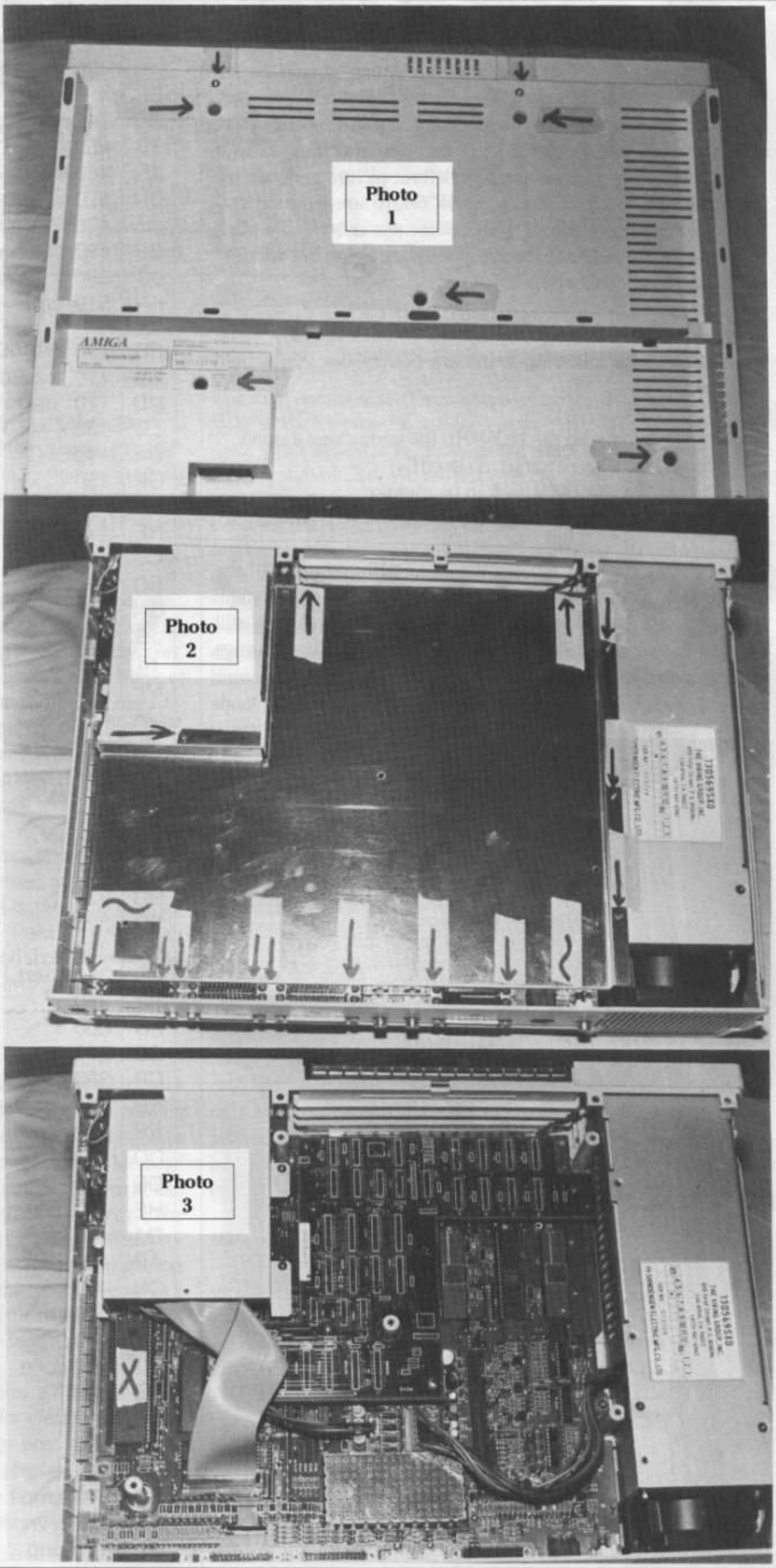
Disconnect all cables from your Amiga and turn the unit upside–down. For this and all the following steps, the rear of the unit should be facing you. To open your Amiga you must remove the 5 recessed screws holding the case together (see Photo 1). Seven screws are marked with arrows. The two at the top of the photo need not be removed to get the Amiga apart. After removing the 5 recessed screws, turn your unit right–side up.

Now comes the trickiest part – opening the case. Examine the seam on the right side of the case: behind the power switch and roughly 10 inches back further are plastic tabs that hold the case together. Push in on these tabs one at a time using a small screwdriver until they release. Now do the same thing on the left side of your case and, after separating all four tabs, remove the top cover. You must now remove the metal RF shielding that covers the entire circuit board. The RF shield has 14 screws securing it in place (see Photo 2). Remove all 14 screws noting where they were removed from, as there are several types of screws used. There are also 2 twisted tabs holding the RF shield in place (see Photo 2, where the tabs are marked with wavy lines). Straighten these tabs out, and remove the RF shield. The MC68000 chip should now be visible (see Photo 3; the MC68000 is marked with an "x").

Using a chip puller, remove the MC68000 and replace it with your new MC68010. It is important to note the correct direction for installation of the MC68010 (see Photo 4). Now reverse all steps to reassemble your Amiga. The metal tabs that helped hold the RF shield do not have to be twisted back into place.

Now it's time to reconnect your Amiga and test the results. For this test you must use V1.1 of Kickstart and Workbench (v1.2 will *not* work for this test, although it is completely compatible with the 68010). Power up the Amiga. Everything should appear to be working in a normal fashion. Now start the Calculator and try "9*9". You should receive a Software Failure; this means that your MC68010 is working correctly. The reason for this error is because the v1.1 Calculator uses the "MOVE SR,ea" instruction, which is now an invalid command. The Calculator supplied with the v1.2 operating system was written with the MC68010 in mind, and does not use the "MOVE SR,ea" instruction. V1.2 will work correctly with the MC68010 and is highly recommended.

Assuming the above sequence has gone well and your results were just as I

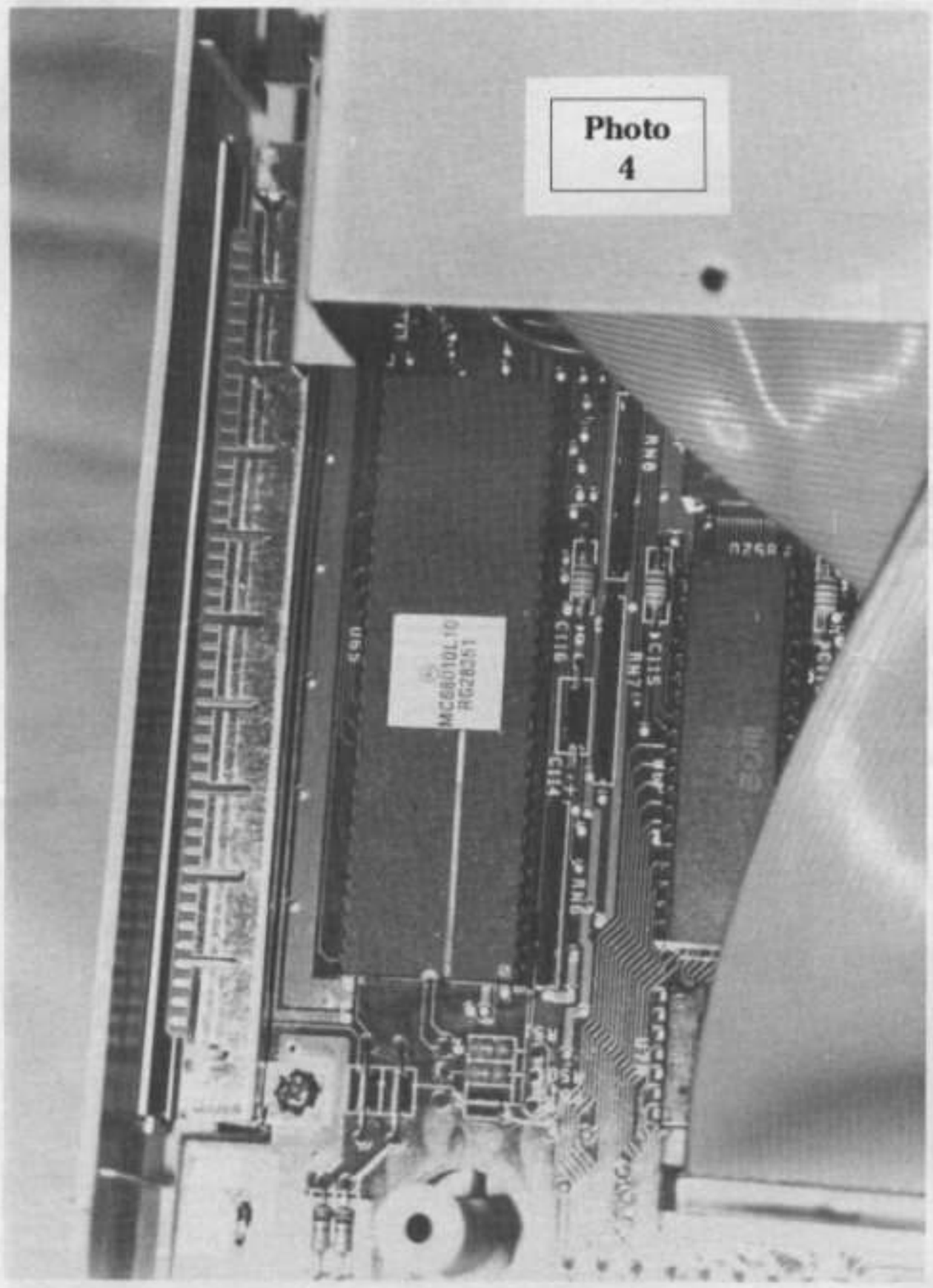


have described, I suggest that you now make the changes listed earlier to your Startup–Sequence. With the MC68010 installed and the DeciGEL wedge running, your Amiga should be between 8% and 50% faster. The speed difference will vary depending on the software you are using. The largest advantage will be noticed when doing a lot of number crunching, as with spreadsheets, ray tracings and Mandelbrot picture generations. To check that DeciGEL and your MC68010 are working correctly together, retry the v1.1 Calculator test described earlier. This time you should not receive a Software Error, but will find out that 9*9 is equal to 81.

If you do not get the expected results, recheck the cables and connections. The following items are possibilities you should check for:

1) You have installed the MC68010 the wrong way around.
2) The MC68010 is dead (i.e. no good)
3) You put the MC68000 back in by mistake
4) You have damaged something else inside your Amiga

I suggest that you first remove the MC68010 and reinstall the MC68000. If your Amiga works with the MC68000 re–installed, then odds are you have a dead MC68010. If your Amiga still does not work I suggest you consult an authorized Amiga service centre for help.



continued from page 49.

```
OF  410        jsr  $ffcc    ;close file
KH  420        jmp  $e37b    ;print ready
EC  430 ;
GE  440 drvnr  =    *
HI  450        .byte 8       ; drive eight or nine
AN  460 initfn =    *
DN  470        .asc "i0:"    ; filename to
JP  480 ;                     initialize drive 0
GC  490 errtxt =    *
JC  500        .asc " evird kcehc"
KN  510 .end
```

```
FD  100 sys 700
NE  110 ;   read the disk error channel
MD  120 ; using the symass 3.13 assembler
LL  130 * = $033c
CA  140 ;
OK  150        lda  #15      ;file number
DM  160        ldx  drvnr    ;device number
NE  170        ldy  #15      ;secondary address
CL  180        jsr  $ffba    ;set parameters
ED  190 ;
GN  200        lda  #0       ;length of filename
JG  210        jsr  $ffbd    ;set filename
KI  220        jsr  $ffc0    ;open file
MF  230 ;
DA  240        ldx  #15      ;#15-error file
IE  250        jsr  $ffc6    ;input from file 15
IC  260        jsr  $ffe4    ;get first char
KP  270        sta  temp1    ;remember it
IM  280        jsr  $ffe4    ;get second char
PA  290        sta  temp2    ;remember it
HL  300        cmp  temp1    ;compare #1 to #2
NG  310        bne  prnterr  ;if not = print error msg
PB  320        and  #$0f     ;mask high nybble
KO  330        bne  prnterr  ;if not zero then print err
FO  340 noerr  jsr  $ffe4    ;no error so continue
EJ  350        cmp  #$0d     ;compare to return
PD  360        bne  noerr
CG  370        jmp  done
CP  380 ;
KB  390 ; print error routine
EP  400 prnterr lda  temp1
DD  410        jsr  $ffd2
DE  420        lda  temp2
HE  430        jsr  $ffd2
DJ  440 loop   jsr  $ffe4    ;get character
AP  450        jsr  $ffd2    ;print to screen
ON  460        cmp  #$0d     ;is it return
GO  470        bne  loop     ;no then loop
GF  480 ;
LB  490 done   jsr  $ffcc    ;reset i/o
AG  500        lda  #15      ;file #15
IN  510        jsr  $ffc3    ;close the file
GK  520        jmp  $e37b    ;basic warm start
FO  530 temp1  nop
BP  540 temp2  nop
KO  550 drvnr  .byte 8       ;either 8 or 9
MA  560 .end
```

The Amiga Section: Messages, Ports and Signals

Chris Zamara, Technical Editor

– Getting tasks to talk to one another is simple!

Introduction

Since the Amiga is a multitasking computer, there are always several programs, known as "tasks" in the jargon of the operating system, operating at any one time. Even if you just run a single program on your Amiga, there are system tasks hard at work taking care of things like the disk drives, keyboard and mouse input, and of course the Workbench, if it has been started.

In order for these system tasks to communicate with the application programs in the system, they use something called a *message*. A message is a chunk of data that is sent to a receiving *message port*, where it can be read by a task. More accurately, it is a chunk of memory in the data segment of one task that can be accessed by another task when the message is "sent" or "put" to a message port. The mechanism for creating message ports and sending messages is provided by Exec, the part of the operating system that handles task–scheduling and other low–level operations.

Messages are not just used as a means of communication between the system and application programs, however; any task in the system can communicate with any other task in the same manner. The C program presented here, "Talking Tasks", provides an example of such inter–task communication, and shows how you can send, receive, and reply to messages in your own programs.

What Messages are Used For

If you have written any programs using Intuition, the Amiga's user interface system, you have already come in contact with reading messages. Input from the user can be read by using Intuition's *IDCMP*. IDCMP stands for Intuition Direct Communication Message Port, and it lets a program find out about events that concern it, like mouse movements, gadget clicks, and keyboard activity. The IDCMP reports these events to a program as messages. The program waits for a message, then reads a message port to get the information contained in the message. Once the program receives a message, it *replies* so that whatever task sent the message knows it was received and can change the message to send a new one.

In the "Talking Tasks" program, we will show a different use of messages in order to illustrate the method for creating your own message ports, finding an existing message port, and sending, receiving and replying to messages. The program itself is not extremely useful other than being a bit of fun, but it shows you how you can write a program that can communicate with other copies of itself that are in the system.

The most common use for message–passing between non–system tasks is when a program spawns a new "child" task, and communicates with this task to tell it things like when to free its resources so that it can be killed. We are not covering spawning a task in this article, but there are still many applications where communication between unrelated tasks can be useful. One interesting example has been suggested by Jim Butterfield: a spelling–checker that runs as a separate task, communicating with the text editor to check words as they are entered. The text editor would just mind its own business, but words would be automatically checked by the spelling checker program that you run on its own. Even if you don't have an immediate application for message–passing, it is important to learn about messages and ports because they are so fundamental to the operation of the Amiga in general.

The Details

Talking about sending a message to a message port sounds like interesting theory, but what does it mean in terms of the computer's real world of bits and memory locations? A message port can be thought of as a place where messages are collected; in real terms it is a "MsgPort" data structure sitting somewhere in memory. In this data structure, among other things, is a pointer to the list of messages that are currently at that port, waiting to be read. There is also a "Node" structure so that Exec can maintain all message ports in a linked list – one of the many lists managed by Exec using the "Node" structure. The system will use the node when you ask it to deal with the MsgPort list, for example when you add, delete, or search for a specific message port. Besides the list node and the pointer to the messages waiting at the port, a MsgPort contains information about the *signals* for that port, which will be explained later.

Here is the definition of a MsgPort structure:

```
struct MsgPort {
  struct Node mp_Node;   /* for system list management use
  UBYTE mp_Flags;        /* defines message–arrival action
  UBYTE mp_SigBit;       /* signal bit number
  struct Task *mp_SigTask;/* task to be signalled
  struct List mp_MsgList;  /* points to linked list of messages
};
```

(For some background about data structures and their use in the Amiga, see the article in the Transactor, Volume 7 Issue 5, "Programming the Amiga".)

Like the message port, the message itself is also a data structure in memory. A message always starts with a "Message" structure, but the message body after that can contain up to 64K of any kind of information, depending on the application. This is the Message structure definition:

```
struct Message {
  struct Node mn_Node;          /* for system list management use */
  struct MsgPort *mn_ReplyPort;/* message reply port             */
  UWORD mn_Length;              /* length of the message in bytes */
};
```

(The MsgPort and Message structure definitions are found in the include file "exec/ports.h".)

You must put a Message structure at the top of any data structure you wish to send as a message. For example, if you wished to send a message containing an (x,y) coordinate, your structure could look like this:

```
struct MyXYmsg {
  struct Message AnyName;
  short Xcoord, Ycoord;
};
```

The "Message" structure is the *system linkage* part of the message, and the information following it is the *body* of the message.

It is important to understand what happens when a message is "sent" to a port. The data is not actually moved from one area of memory to another; the port simply gets a *pointer* to the message data. In other words, messages are passed by reference, not by value. You can think of a message as a temporary licence for another task to use a space in the data segment of your task – you put the information you want in the message, then allow the other task access to that information by sending the message.

Since both the task sending the message and the task receiving the message have access to the same data at the same time, it is important that the sending task doesn't change the contents of the message structure while the receiving task is trying to read it. This is where message replying comes in. After sending a message, a program generally waits for a *reply* from the task that received it. A reply is just a message that the receiving task sends to a designated port, called the reply port (the reply port can be the same port that received the original message). After sending the message, the sender should not modify the contents of the message structure until it gets a reply; at that time, the receiving task has finished with the message and should no longer try to access data within it. The receiving task, on the other hand, may change the message *before* it replies so that it can send new information back to the sender.

To make things a little more concrete, here are algorithms that could be used to send and receive messages.

In order to send a message to a specific port:

1) Create a port • port = CreatePort(name, priority)
   or get a pointer to an existing port • port = FindPort(name)
2) Put desired data in a message structure
3) Send the message to the port • PutMsg(port, message)
4) (optional) Wait for the reply • WaitPort(ReplyPort)
5) (optional) read data in reply message • message = GetMsg(ReplyPort)

In order to receive a message from a specific port:

1) Create a port • port = CreatePort(name, priority)
   or get a pointer to an existing port • port = FindPort(name)
2) Wait for a message to arrive at the port • WaitPort(port)
3) Get a message from the port • message = GetMsg(port)
4) Read data of interest in the message
5) (optional) Reply to the message • ReplyMsg(message)
6) Do whatever action is dictated by the message data
7) Repeat 3 through 6 until there are no more messages

As you can see, the code required to send and receive messages is not all that complicated, but you can get into trouble if you're not careful. Remember that the data in a message should only be changed by the task that currently has "ownership" of the message. The task sending the message has ownership at all times except after sending the message and before receiving the reply. The task receiving the message has ownership only after getting the message and before replying. After replying to a message, you can make no assumptions about the validity of data in the message you've just received. A common mistake is something like:

```
msg = GetMsg(SomePort);   /* get the message   */
ReplyMsg(SomePort);       /* reply to the sender */
x = msg->something;       /* uh-oh! Bad news!  */
```

The correct approach would be to reverse the second and third lines of code so that the desired data was fetched from the message *before* the reply was given. Once you reply, do not assume that the message still holds valid data – after the other task got the reply, it may have changed the data in the message, or released the memory used by the message back to the system.

The Functions Used

Let's take a look at the functions that were introduced briefly in the above section.

The CreatePort( ) function is not actually in the ROM kernal itself, but is a short "exec support" routine that is in the library "amiga.lib" ("c.lib" with the Aztec C compiler) and is linked with your program. It takes as arguments the name (a pointer to a string, or zero) and the priority of the port to be created (zero is normally used). It returns a pointer to the port that it creates (a pointer to a MsgPort structure). The function allocates a signal bit for the port (more on signals later), allocates memory for a MsgPort structure, initializes various fields in the MsgPort, and if the name given was not NULL, adds the port to the system with the AddPort( ) function so that other tasks can access it – this is called a *public port*. Unless both communicating tasks have a pointer to the message port being used (as is the case with Intuition and an application program), a port should be made public so that any task can use it just by knowing its name. You

should try to ensure that the name you give to a public port is unique so that there will be no conflicts with other tasks.

All ports created with CreatePort( ) must be deleted with DeletePort( ) before the task ends. DeletePort takes a pointer to the port to be deleted as its only argument.

FindPort( ) returns a pointer to a port, given that port's name. If no port with the given name can be found, it returns NULL (zero). Using FindPort( ), you can send messages to a port created by another task, as long as you know the port name. This only works with public ports (those given a name when created with CreatePort( ), or added with AddPort( )).

PutMsg( ) sends a message to a port. It takes as arguments a pointer to the port and a pointer to the message, respectively.

GetMsg( ) gets a message – if any – from the given port and returns either a pointer to the message, or NULL if there are no messages at the port. To get all messages from a port, you should call GetMsg( ) until it returns NULL.

WaitPort( ) waits for a message to arrive at a given port, putting the task to "sleep" until one arrives. A sleeping task uses no CPU time. It returns a pointer to the first message to arrive at the port, but there may be more than one message at the port after WaitPort( ) returns. You should get all messages at the port after a WaitPort( ), as described above. WaitPort( ) does NOT remove the message from the port, so you *must* do a GetMsg( ) afterwards to remove it.

ReplyMsg( ) sends the given message to its "Reply Port", a pointer to which is contained in the Message structure. If you wish to use ReplyMsg( ), the 'mn_ReplyPort' field of the message structure must contain a pointer to a port; the pointer is normally put there when the message is prepared before it is sent. The same port that the message was sent to may be used as the reply port.

Signals

Just in case you feel disappointed because this topic is too simple for you to work your brain around, here's some more nourishment for cerebral satisfaction. You don't *have* to know all about signals to pass messages as described above, but this section might answer a few questions that have arisen (and probably create as many new ones!).

Signals are used to "wake up" a "sleeping" task. Each task has up to 32 signal bits that it can use, and it can wait for any of these signals to occur by using Exec's Wait( ) function. When a task is waiting for a signal it uses no CPU time, so signals allow many tasks to be active in the system at once, waiting for user input or other external events, without slowing down other, hard–working tasks.

As you have probably guessed, signals play an important role in message passing. When a task is waiting for a message (or a reply to a message, which is no different), it is really waiting for a given signal to occur.

For a better understanding of how signals relate to messages and ports, let's take another look at the MsgPort structure, specifically the fields called 'mp_Flags', 'mp_SigBit', and 'mp_SigTask'.

These fields are filled in when you create a port with the exec support function CreatePort( ). Exec uses the information found in these fields in a port to determine what action to take when a message arrives at that port. CreatePort( ) sets up the fields so that your task is signalled when a message arrives at the port; You can set up a port structure yourself and use AddPort( ) instead of CreatePort( ) if you aren't going to use WaitPort( ) or Wait( ) to wait for a message to arrive at the port, and thus don't care about getting signalled.

Let's look at the fields one at a time:

```
UBYTE mp_Flags;    /* defines message-arrival action */
```

Depending on the value in 'mp_Flags', Exec will do one of three things when a message arrives at the port: generate a signal, generate a software interrupt (software interrupts will not be mentioned again in this article), or do nothing. The values corresponding to these actions are the constants PA_SIGNAL, PA_SOFTINT, and PA_IGNORE, respectively (these are also defined in the include file "exec/ports.h"). The CreatePort( ) function uses PA_SIGNAL so that your task can sleep while waiting for a message to arrive at the port.

```
UBYTE mp_SigBit;   /* signal bit number              */
```

We mentioned the fact that there are 32 signals that any given task can wait for. The 'mp_SigBit' field gives the number of the signal bit that is set when a message arrives at the port (if the PA_SIGNAL option is used). CreatePort( ) allocates a signal bit and puts the newly–allocated signal bit's number in this field.

```
struct Task *mp_SigTask;        /* task to be signalled */
```

Every task has its own set of signals. The 'mp_SigTask' field is a pointer to the task that is to be signalled when a message arrives at the port (again, this only applies if the PA_SIGNAL option is chosen). CreatePort( ) fills this in with a pointer to the task that is currently running, in other words, the task to signal is "me", or "this task".

Waiting for Signals

Again, the above information is not necessary if you just want to pass messages. Knowledge about the 'mp_SigBit' field, however, is useful if you want a program to wait for messages arriving at more than one port.

We have discussed the WaitPort( ) function, which will put your task to sleep until a message arrives at the given port. An alternative is the Wait( ) function, which allows you to wait for one or more signals to occur. Wait( ) takes as an argument a bit–mask defining which signals to wait for.

Since your task will receive a signal when a message arrives at a port, and since the number of the signal is indicated by the mp_SigBit field, you can use Wait( ) to wait on the signal instead

of WaitPort( ) to wait on the port. For example, to wait for a message to arrive at the port "Smurf" (the name of a pointer to a MsgPort structure), you could use:

```
WaitPort(Smurf);
```

or,

```
Wait(1L << Smurf->mp_SigBit);
```

The latter statement has the same effect, but takes the signal bit as an argument instead of a pointer to the port. The advantage is that you can wait on several ports simply by using the bitwise OR of the signal bits in each port. For example, to wait for a message arriving at any of the ports "Smurf", "ThisPort", or "ThatPort":

```
Wait ((1L << Smurf->mp_SigBit) |
      (1L << ThisPort->mp_SigBit) |
      (1L << ThatPort->mp_SigBit))
```

You could then get any messages from all three ports and handle them in the usual way.

The Talking Tasks Program

Now comes the practical (sort of) application of the concepts we've covered. The program that appears in the accompanying listing puts everything together and illustrates message passing for both the programmer studying the source code and the user playing with the program itself.

Talking Tasks, as it is called, lets you create a community of tasks, each with its own name, and each with its own window. Each copy of the Talking Tasks program that is running has the ability to "talk" with any other copy by sending typed messages. Also, all talking tasks have access to a public message port called "Joe's Cafe", where they can discover how many talking tasks are currently running and use this information to determine the best place to put their window so that all windows will come up in a different location without having to be dragged around by the user.

To use the program, enter, compile and link the C program listed, or otherwise get an executable copy of the program on disk. (Transactor programs tend to spread around, thanks to our public-domain software policy.) This version was compiled with Manx Aztec C v3.40a – it will most likely work with Lattice, but hasn't been tested with that compiler. Also, it works with V1.2 Kickstart/WorkBench; it should also work with V1.1 but it hasn't been tested with that version. The executable version of the program should be called "ttalk" on disk for quick typing.

Ttalk should be run from the CLI using the RUN command so that the CLI will still be available to run more ttalks. Ttalk takes one argument, which is the name you would like to give that talking task. As an example, to create a small community of four talking tasks, you could type the following commands from CLI (make sure the "ttalk" program is in your current directory).

```
run ttalk Ernie
run ttalk Edna
run ttalk Jimmy
run ttalk Bertha
```

(You'll have to click on the CLI window after each RUN to re-activate it.)

Notice that the small window for each program comes up in a different place, even though you are running the same program each time – your first indication that the tasks are indeed talking with each other in some way.

Now, let's say you want Ernie to talk with Edna. Activate Ernie's window (his name will be in the window's title bar) by clicking with the mouse, then address Edna and type your message, something like this:

```
Edna,Gee you look lovely today.
```

You will see within Edna's window that she did indeed receive this flattering message from Ernie, and the message printed in Ernie's window indicates that Edna acknowledged the message – Ernie knows that his kind words were not falling on deaf ears. Edna can now send a message back to Ernie, or anyone can send a message to anyone. For example:

```
(click in Edna's window)    Ernie,Get lost, creep!
(click in Jimmy's window)   Ernie,Hi pal. How's life?
(click in Ernie's window)   Bertha,Hey baby, love your nails.
(click in Bertha's window)  Ernie,You Scorpios are all alike!
```

Well, it's not impossible that eventually this great fun may wear out, but the point is clearly made: separate tasks are getting messages across to each other, and are also sharing some common data to determine where to put their window. While you're having fun, try sending a message to someone who doesn't exist (don't worry, it's safe); try giving talking tasks duplicate names; try sending a message to "Joe's Cafe"; sending a message to "yourself".

To kill a talking task, just press return without entering any text.

How It Works

There are two kinds of communication going on among the talking tasks: the direct sending of messages from one task to another, and the sharing of information by all talking tasks at the port called "Joe's Cafe".

Joe's Cafe is used so that when a talking task is first run, it can find out how many others are already running and can use this information to decide on a reasonable place to put its window. The function HowMany( ) determines the number as follows: It looks for Joe's Cafe using FindPort( ). If it is not found, it creates Joe's Cafe by allocating space for a MsgPort structure, filling it in, then using AddPort( ) to make it public. There is no need to use CreatePort( ) in this instance, since a signal is not needed for the port, as there will be no waiting for a message to arrive. A message is then created that contains the mandatory "Message" structure followed by an integer used to store the number of talking tasks currently running. This count is initialized to zero, and the message is put to the Joe's Cafe port with PutMsg( ).

If Joe's Cafe already exists (it was found with FindPort( )), then the message is read with GetMsg( ), the count in the message is

looked up and incremented, and the message with the new count is put back to the port for the next talking tasks to look at. Just as the first talking task to be run creates Joe's Cafe, the last one to exit removes it. The port is called Joe's Cafe because it is a place where all talking tasks hang out.

While only the first task to be run creates the Joe's Cafe port, every task creates a port of its own, using the name given by the user when the program was run from the CLI. So, the name of Edna's port is "Edna", and now you can probably guess the port names of the characters in any given talking task community. This port will be used to receive messages sent by other talking tasks, and to receive replies to messages sent to others. Unlike Joe's Cafe, this port is created with the CreatePort( ) function, since it not only needs to be public, but it also needs to use a signal so that WaitPort( ) can be used to wait for replies.

After HowMany( ) has been called to find the number of talking tasks currently living in the community and the message port has been created, a DOS file is opened to create the window used for text input and output. The filename is picked based on the number of tasks there are, so that the window will be in a different place every time the program is run, repeating positions every six times (each window takes up one sixth of a 640 by 200 screen).

Now comes the hard part: the program must wait for input typed into the window by the user AND wait for any messages arriving at the port. If WaitPort( ) were used to wait for messages, user input would be ignored, and if Read( ) were used to get input, messages would be ignored. A simple approach is used to solve this problem: the DOS function WaitForChar( ) is used to wait up to a tenth of a second for a character to be typed by the user. If a character is typed, Read( ) is called to read the text entered by the user, and the text is passed to a function called SendString( ) to be processed. Whether text was entered within the 1/10–second time frame or not, the message port is checked for messages with GetMsg( ). Messages are read until no more are found at the port. For each message read from the port, the name of the sender and the text sent are read from the message body and printed to the window for the user to see. This loop repeats continuously until a single newline is typed, ending the program.

This might not be the most high–performance way to read user input and messages from a port, but it works well and has no noticeable effect on system performance. If you wish, the 1/10 second time can be increased (by increasing the value passed to WaitForChar( )) to make less demand on the CPU when the program is sitting idle.

When something is typed by the user, the string is passed to SendString( ), which splits it into the name (the text before the first comma) and the message, then calls SendMessage( ). SendMessage( ) looks for the port with the given name using FindPort( ). If the port is not found an error message is printed, and if it is found, a message is prepared and sent to the port. The kind of message structure used is called "MyMessage" and is defined near the beginning of the program listing. A "MyMessage" structure has the usual "Message" structure at the top, then has two pointers to strings: the name of the sender of the message, and the text that is to be sent. An instance of a MyMessage structure simply called "message" is declared in the SendMessage( ) function. This structure is filled in with the proper data, then sent to the previously–found port using PutMsg( ). WaitPort( ) is then used to wait for a reply, and the reply message is removed from the reply port with GetMsg( ). A line of text is printed to the window telling the user that the reply was received.

That's all there is to it. Look at the program listing to get a better idea of how everything's done; the code is easy to understand and well–commented. With the program as a sample and this article, you should have no trouble using messages and ports in your own programs. Not only will you have new ways to solve programming problems, but you will have learned about a fundamental mechanism within the Amiga's system software, and will come one step closer to mastering the machine.

```
/* "TTalk" - Talking Tasks
 * - programming example using messages and ports
 * (C) 1987 Transactor Publishing Inc.
 * From Transactor Magazine, written By Chris Zamara, May 1987
 * ---->> This program may be freely distributed <<----
 *
 * This code shows you how to create and find message ports, and how
 * to send, receive, and reply to messages.
 *
 * This program, "TTalk", lets you create several named DOS windows, and send
 * messages between them. Just give each task its name when you run it
 * from the CLI, e.g. "run TTalk Fred". If you then start another task,
 * like "run TTalk Edna", you can send Edna a message from Fred by typing
 * into Fred's window something like, "Edna, you look lovely today!"
 * Edna will receive the message and print it in her window. Any number
 * of these tasks can be started, and any one can talk to any other.
 *
 * How it works: A message port is created and given the name that the user
 * supplies (the name in the window title). To send a message to another task,
 * its port is found with FindPort(), a message is sent to the port with
 * PutMsg(), and a reply is waited for with WaitPort(). Within the message is
 * a pointer to the text that the user wanted to send. The receiving task
 * uses GetMsg() in between waiting for keypresses to receive the message.
 * All talking tasks also have access to a public port called "Joe's Cafe",
 * where they read a message saying how many talking tasks are running, and
 * update the message when they are started and ended. By talking at Joe's
 * Cafe, the talking tasks can determine a good place to put their window so
 * that the user doesn't have to always move around overlapping windows. The
 * first task started creates the "Joe's Cafe" port and the first message
 * there, and the last task ended deletes them.
 *
 * compiled with Manx Aztec 3.40a, should work with Lattice as well.
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <exec/ports.h>
#include <libraries/dos.h>

/* Print macro used to send a string to the output window */
#define Print(s) Write(IOfile, (s), (long)strlen(s) )

#define BUFLEN 200 /* length of input buffer for user text entry */

/* this is the structure for the message we will be sending */
struct MyMessage {
  struct Message Msg;      /* for Exec message routines  */
  char *NameOfSender;   /* sender puts his name here  */
```

```
  char *text;                   /* the text we want to send      */
};

/* this is the kind of message we will use to
 * determine how many talking tasks are currently running
*/
struct CountMsg {
  struct Message Msg;
  int Count;
};

/* external function declarations */
extern BPTR  Open();
extern ULONG Read();
extern UBYTE *AllocMem();
extern struct MsgPort *CreatePort(), *FindPort();
extern struct MyMessage *GetMsg();

/* global variables */
char *MyName;                   /* ptr to name given to this 'talking task'  */
struct MsgPort *MyPort = NULL;  /* message port for sending/receiving msgs */
struct MsgPort *TTport;         /* port shared by all Talking Tasks          */
char *TTportName = ''Joe's Cafe'';/* name of TTport, where they all hang out */
struct CountMsg *TTmsg;         /* the message we'll leave at Joe's Cafe     */
BPTR IOfile = NULL;             /* DOS file handle for the terminal window   */


/***** start of main() ************************************************/
main (argc, argv)
int argc;
char **argv;
{
 /* give user instructions and exit if invalid args passed */
 if (argc != 2 || strlen(argv[1]) > 30)
 {
   printf(''Run me with a name, like: 'run %s Ernie'\n'', argv[0]);
   exit(0);
 }
 MyName = argv[1]; /* first argument is name given to this 'talking task' */
 /* open DOS window, create ports, etc. and return TRUE if successful    */
 if ( OpenStuff() )
   HandleInput(); /* get input, send and read messages until user exits  */
 CloseStuff();
}


/* OpenStuff() ********************************************************
 * create 'MyPort' message port, call HowMany() and open DOS window
*/
OpenStuff ()
{
static char windowName[50]; /* holds filename for DOS ''con:'' window */
/* this array is used to choose an appropriate window position */
static char *conNames[] = { ''con:0/0/319/65/'' , ''con:320/0/319/65/'',
                            ''con:0/67/319/65/'' , ''con:320/67/319/65/'',
                            ''con:0/134/319/65/'', ''con:320/134/319/65/''
                          };
 /* see if a message port with the given name already exists */
 if (FindPort(MyName))
 {
   printf(''Hey, there's already someone here called '%s'!\n'', MyName);
   return (int)FALSE;
 }
 /* set up a message port with the given name and get a pointer to it */
 MyPort = CreatePort(MyName, 0L);
 if (MyPort == NULL)
 {
   printf(''can't open '%s' port!\n'', MyName);
   return (int)FALSE;
 }
 /* the number of talking tasks running determines where to put the window */
 strcpy(windowName, conNames[HowMany() % 6]);
 strcat(windowName, MyName); /* 'MyName' is title for DOS window */
 IOfile = Open(windowName, MODE_NEWFILE); /* open DOS window for user
I/O */
 if (IOfile == NULL) /* file didn't open for some reason */
   return (int)FALSE;
 /* print some instructions */
 Print('' '(send message with <name,message. . .>)' '');

 return (int)TRUE; /* everything opened OK */
}

/* CloseStuff() *******************************************************
 * Undo what OpenStuff() and HowMany() did
*/
CloseStuff ()
{
 if (IOfile)
   Close(IOfile); /* close DOS window if open */
 /* decrease count in TTport message, remove port if count is zero */
 if (TTport = FindPort(TTportName))
 {
   TTmsg = (struct CountMsg *)GetMsg(TTport);
   if (TTmsg->Count--)  /* still more talking tasks, don't remove port */
     PutMsg(TTport, TTmsg); /* put back message with decreased count */
   else
   { /* we're the last talking task, remove TTport and TTmsg  */
     RemPort(TTport);  /* remove the port  */
     FreeMem(TTmsg, (ULONG)sizeof(*TTmsg));
     FreeMem(TTport->mp_Node.ln_Name, (ULONG)(strlen(TTportName) + 1));
     FreeMem(TTport, (ULONG)sizeof(*TTport));
   }
 }
 if (MyPort)
   DeletePort(MyPort);     /* delete our main message port */
}


/* HandleInput() ******************************************************
 * Get user input from window and any messages arriving at MyPort.
 * Text from the user is passed to SendMessage().
 * Print text field of incoming messages, then reply to message.
 * Returns when user inputs a null text line.
*/
HandleInput ()
{
char InputBuffer[BUFLEN];
BOOL exit_flag = FALSE;
struct MyMessage *msg;

 /* We want to get keyboard input from the user AND get messages arriving
  * at our message port, and we don't want to waste much CPU time.
  * So we WaitForChar() and if no character is received within 1/10
  * second, we read the message port, process any messages there, and try
  * again. This way we only GetMsg() every 1/10 second, which is cheap.
 */
 while (exit_flag == FALSE)
 {
   if (WaitForChar(IOfile, 100000)) /* wait up to 1/10 second (in micros) */
   {
     /* read an input line and send the message */
     if (Read(IOfile, InputBuffer, (long)BUFLEN) > 1)
       SendString(InputBuffer);
     else
```

```
        exit_flag = TRUE; /* newline by itself means user exit */
    }
    /* now handle any messages for us at the port */
    while (msg = GetMsg(MyPort)) /* loop until all messages processed */
    {
      Print(' 'A message from ' ');
      Print(msg->NameOfSender);
      Print(' ': \n \' '' ');
      Print(msg->text);
      Print(' '\' '\n' ');
      /* We took care of the message, now reply to it. */
      ReplyMsg(msg);
    }
  }
}


/* SendString(text) ************************************************
 * Split the given string into two strings at first comma
 * and call SendMessage( ) with the resultant strings.
 * Print error message if no comma found.
*/
SendString (text)
char *text;
{
int NamePos;

 NamePos = SearchChar(text, ',', BUFLEN); /* check for comma */
 if (NamePos = = BUFLEN) /* no comma */
   Print(' '(send message with <name,message. . .>)' ');
 else
 { /* split string into two and give strings to SendMessage( ) */
   text[NamePos] = '\0';
   text[SearchChar(text, '\n', BUFLEN)] = '\0';
   SendMessage(text, text + NamePos + 1);
 }
}


/* SendMessage(name, msgstring) ************************************
 * Given a port name and a text string, find the port and send a
 * message containing the string to it, and wait for a reply.
*/
SendMessage (name, msgstring)
char *name, *msgstring;
{
struct MsgPort *HisPort;
struct MyMessage message;

 HisPort = FindPort(name); /* look for other fellow's message port */
 if (HisPort = = NULL)     /* NULL means port couldn't be found   */
 {
   Print(' 'Can't find ' ');
   Print(name);
   Print(' '!\n' ');
 }
 /* error if message being sent to ourselves */
 else if (strcmp(name, MyName) = = 0)
   Print(' 'Talking to myself. . . OK!\n' ');
 else if (strcmp(name, TTportName) = = 0)
 { /* don't send to Joe's cafe!! */
   Print(' 'Oh no you don't!\nHumans aren't allowed at ' ');
   Print(TTportName);
   Print(' '.\n' ');
 }
 else /* everything's OK, prepare the message and send it to his port */
 {
   message.Msg.mn_Node.ln_Type = NT_MESSAGE; /* for Exec list handling    */
   message.Msg.mn_Length = sizeof(message);  /* number of bytes in msg */
   message.Msg.mn_ReplyPort = MyPort;        /* so receiver can reply    */
   message.NameOfSender = MyName;            /* tell him who sent it     */
   message.text = msgstring;                 /* our text string to send  */
   PutMsg(HisPort, &message);                /* send the message         */
   WaitPort(MyPort);                         /* wait for a reply         */
   GetMsg(MyPort);                           /* remove reply from port */
   Print(' ' <Got acknowledgement from ' ');  /* tell user we got reply   */
   Print(name);
   Print(' '>\n' ');
 }
}


/* SearchChar(string, chr, n) **************************************
 * find character 'chr' in 'string', searching up to n characters
 * return n if character not found
*/
SearchChar (string, chr, n)
char *string;
int chr, n;
{
int i;

 for (i = 0; i < n && string[i] != chr; i++)
   ;
 return (i);
}


/* HowMany( ) ******************************************************
 * Determines how many 'talking tasks' are currently in the system. Look for
 * a port named TTportName (Joe's Cafe). If it exists, get a 'CountMsg'
 * message from it, read the count, increment it and put the message back. If
 * the port doesn't exist, create it and put a message there with the count
 * field set to zero. Return the value of the count.
*/
HowMany ( )
{
int count;

 if (TTport = FindPort(TTportName) )
 {
   TTmsg = (struct CountMsg *)GetMsg(TTport); /* get message. . .  */
   count = ++TTmsg->Count;                    /* bump count. . .  */
   PutMsg(TTport, TTmsg);                     /* and put it back  */
 }
 else
 { /* port not there, we are first talking task - create the port */
   TTport = (struct MsgPort *)AllocMem( (ULONG)sizeof(*TTport), MEMF_PUBLIC);
   TTport->mp_Node.ln_Name = (char *)
     AllocMem( (ULONG)(strlen(TTportName) + 1), MEMF_PUBLIC);
   strcpy(TTport->mp_Node.ln_Name, TTportName);
   TTport->mp_Node.ln_Pri = 0;
   TTport->mp_Node.ln_Type = NT_MSGPORT;
   TTport->mp_Flags = PA_IGNORE;
   AddPort(TTport); /* make the port public so all tasks have access */

   /* now create the message to put in the port (Joe's Cafe) */
   TTmsg = (struct CountMsg *)AllocMem( (ULONG)sizeof(*TTmsg), MEMF_PUBLIC);
   TTmsg->Msg.mn_Node.ln_Type = NT_MESSAGE; /* for Exec list handling    */
   TTmsg->Msg.mn_Length = sizeof(*TTmsg);   /* number of bytes in msg */
   TTmsg->Msg.mn_ReplyPort = NULL;          /* no reply port required   */
   TTmsg->Count = count = 0;                /* start count at zero      */
   PutMsg(TTport, TTmsg);          /* leave a message at Joe's Cafe for everyone */
 }
 return count;
}
```

Amiga Dispatches

by Tim Grantham, Toronto, Ontario

Rattigan wrongigan?
Shepherd led the flock astray?
Reign of Error over?

Such were the lurid headlines that flashed through my mind when I heard the news of the latest corporate spasm at CBM. I mean, really, it's getting to be worse than *Dallas*. Big Daddy Gould ('Irving' to his friends and detractors) stepped in and forced CEO Thomas Rattigan, who had just had his contract renewed for five years, to resign after having had him physically escorted from Commodore HQ. Vice–President Nigel Shepherd was also dismissed.

The official reason was that Rattigan had let the US market deteriorate to the point where West Germany had become CBM's biggest source of customers. That he had also overseen CBM's return from the brink of bankruptcy to profitability in the last three quarters apparently counted for naught.

Commodore's European and Canadian operations have almost always done well — it's the US that has proved to be their Achilles heel. This may be the rationale behind Rattigan's and Shepherd's replacements: Alfred Duncan and Richard McIntyre. Each has been a chief of CBM Canada and Duncan also headed up the Italian branch. With Gould acting as CEO, it is hoped that they will help CBM find the golden touch once more in the US.

Rattigan is suing CBM for $9 million, the claimed worth of his now defunct contract. CBM's stock has dropped in price from $12 to $10.

The recent change in the announced prices of the Amiga 500 and 2000 may be the result of this management shakeup. The 500 has increased $50 (US) to $700, $200 more if you want 1 Mb, and the 2000 has jumped a whopping $500 to $2000.

Naturally, I'm disappointed in the higher price of the 2000. However, I think it may be a smart thinking on the part of CBM, for several reasons. One, those who have the need and the bucks for hard drives, bridge cards, 68020/68881 cards, et cetera, that can be easily added to the 2000, will not balk at paying an extra $500 -– they're still getting a very high performance machine for less than two grand. Second, and more important from a strategic viewpoint, it will make it more attractive to dealers. I suspect a large portion of that extra $500 will be passed on as markup. Not only does it give dealers more room for competitive pricing, it provides the revenue they will need to pay for the support buyers of the 2000 will demand. If CBM is indeed serious about rehabilitating their dealer network in the US, they will also improve their technical support, their advertising and their product availability.

The new pricing also evens out the Amiga product profile, placing the 500 where it can still compete with the Atari ST as a home computer, the 2000 where it can provide the power required by the business and the professional user, and the 1000 squarely in the middle as an advanced personal computer that can still fulfill it's initial promise as a tool for creative computerists.

Meanwhile, in–house development has not come to a standstill. 1.3 of the Amiga OS is being written and is reported to consist mostly of further optimization and the ability to boot from a hard drive (a new boot ROM might need to be installed for this). Don't expect to see it soon. Tim King, the author of AmigaDOS, is apparently busy rewriting the BCPL code into assembly language, making it faster and easier to interface with. A custom memory management unit for the 2000, required to run UNIX, is being put into silicon as you read this. Last, and not least: although CBM shut down their Commodore–Amiga headquarters in Los Gatos and terminated all but two of the employees, they have opened a small C–A office in that city. There, Bart Whitebook, Manager of Amiga ROM Software, has been joined by Dale Luck and Carol Havis, Manager of Third Party Software (and also spouse of R. J. Mical and mother of Alexander Jose Mical). They are charged with maintaining, enhancing and developing the ROM software. It's good to see that CBM has not completely shut down R&D.

Shareware and the Public Domain

There is no doubt in my mind that the very high quality of Shareware and PD software for the Amiga has contributed significantly to its survival. The Fish collection, the Amicus library, even our first Transactor Amiga disk, have filled in the gap between expensive commercial offerings and homebrew AmigaBasic programs. They have provided not only useful software at low cost but absolutely invaluable programming info in the (usually) accompanying source code.

However, I don't think this will continue, at least not at the same level. The unfortunate reality is that those who have relied on the honesty of others are not getting their just reward. Authors like Hayes Haugen (**Blitz!, Blitzfonts**), Rick Stiles (**Uedit**) and the Software Distillery (**PopCLI, Blink, Hack, Larn**) have seen little return on the faith they have placed in the Amiga user. If you want 'em, folks, ya gotta pay for 'em. TANSTAAFL.

My personal favourites out of the current crop, besides the aforementioned ones, are **conman**, by William S. Hawes, and Steve Drew's **aux2**.

The former slips into the CON: device and provides a command history and line editing. Any program thereafter that uses the CON: device will also have these features. It's not nearly as powerful as Matt Dillon's **csh** C shell, but it takes no time to learn, uses a miniscule amount of memory and I couldn't live without it.

aux2 sets up an AUX: device on the serial port. Another user can connect a terminal, either directly or via modems, and operate the Amiga remotely on a CLI opened on the serial port — thus fulfilling, within limitations, the original promise of the Amiga as a multi-user machine. You will *not* get Amiga graphics on the terminal's screen. For example, you can call up a new CLI by entering the **newcli** command, but the new window will be appear on the host Amiga. CLI's only deal with character data. Only those programs that use stdin, stdout and stderr will work directly with the terminal.

So what's it good for? Currently, not a great deal. One could probably play text-only adventure games remotely. You could compile a C program remotely — though all your source, include and link files would have to be on the host machine. (If your terminal program had a buffer send function, you could upload your source code by entering **copy * to df1:test.c**). You could snoop in your friend's files, or your own when you're on a trip. But until there are Amiga programs that send to/receive from a terminal-type device, it's not going to be of great use.

Nevertheless, it works, it has potential and it's impressive. Anybody out there have other ideas for it? Let me know.

(Michael Rosenberg, of Conceptual Computing here in Toronto, has written a much more elaborate system that combines several modules to permit an unlimited number of terminals to be hung on the serial port, via a multiplexer. Each terminal can open multiple windows and run multiple tasks. Each 'window' on a terminal consists of a screen; they cannot be resized. They can, however, be pushed behind each other. The programs run must be text-based, but Rosenberg's software permits input/output redirection between programs and more sophisticated editing capabilities than **aux2**. **Ed**, for example, can be run remotely on multiple terminals, providing enough memory is available on the host Amiga. Rosenberg expects that most of those interested in his system will be writing their own application software. The complete package costs $150 (Can.) For more info, you can reach Rosenberg at (416) 781-7742.)

Not quite in the headline category are demo versions of commercial software. These can still be useful, even though they are usually crippled to some extent. **Disk2Disk**, for example, is from Central Coast Software, the same people who wrote **Dos2Dos**. **Disk2Disk** can transfer files between AmigaDOS disks and 1541/71 disks for the the C64/128. Note that it *cannot* run these programs — you would need a C64 emulator for that. But you can copy, read, list. . . There is also a feature that will check incompatibilities between Commodore BASIC programs and AmigaBasic programs. I've been able to use it to remind myself of all the things I could do on my C64. It costs $49.95 (US).

And now, hot off the nets. . .

ACO, developed by a team headed by Steve Pietrowicz (CBM STEVE of PeopleLink's Amiga Zone), in an Amiga online conferencing program inspired by VMCO for the Mac. It can be downloaded from Plink. . . Mike Plitkins and Ralph Navarro of Top Disk Software are hard at work on a CP/M emulator. . . There are new versions of Electronic Arts' **Deluxe Music Construction Set** and **Deluxe Video Construction Set** out. Bugs have been reported, particularly when using them with hard disks. EA is following them up with fixes. . .

Data Pacific is working on an external disk drive that attaches to the serial port to read Mac disks. . . Look for **DesignText**, an $80 (US) word processor from Brian Niessen's group in Vancouver. . . **PageSetter** appears to be getting good response from its owners. They now have a PostScript utility available and have made their **PagePrint** freely copyable; you can expect to see user group newletters being distributed electronically in **PageSetter** format. . . **Morerows** is a neat little PD program that enlarges the available screen area for printing. You can now have a full 80 columns in your CLI. . . **Word Perfect** is finally due to arrive this summer. It won't be cheap, but it will be powerful. Response from beta testers has been very positive. . . **Starglider**, by England's Jez San, has crossed the ocean at last. . .

Oxxi has a new version of **MaxiPlan Plus** ready and Marco Papa has introduced **A-Talk Plus** which, among other things, adds Tektronix terminal emulation. . . Joe Lowery of New York is busy organizing AmiExpo, which will take place October 10-12, 1987 at the Sheraton Hotel in the Big Apple. It will be followed by an LA version Jan. 22-24 and a Chicago appearance July 22-24, both in 1988. . . Excellent Amiga book: *Programmer's Guide to the Amiga*, by Rob Peck, a member of the original Los Gatos development team. It's published by Sybex.

Amiga-Tax ($59.95) is a Canadian income tax calculation program from Imagec Software Productions. I used it to check my own return. It did everything perfectly, until it got to Schedule 1, the detailed tax calculation. Here, unfortunately, it inverted two numbers on the second line, the end result being that it said I owed *less* tax than I had paid. In addition, my copy had a major bug in the tax records forms. This appears to be a well thought-out, carefully implemented program. However, those who tackle this kind of task have to be extra careful. If it can't be trusted right from the word go, it won't be bought.

The new version of Online! (2.00) adds Kermit protocol and autochop, among other features. Owners of earlier versions can upgrade for a small fee. Note that publishers Micro Systems Software have changed their address. Their new technical support number is (305) 790-0772.

Perhaps they'll be able to help me. I have the new version of **Scribble!**, the word processor for the Amiga. It has a number of improvements, but has taken a step backwards in its support of multiple windows. In the previous version, I could have a full four windows open, although sometimes there apparently wasn't enough memory for the directory requester. Now it's totally inconsistent: I can call up a 100k buffer from the CLI, but not from within the program. I managed to get two 64K buffers open, but no directory requesters. I opened a 30k buffer and a 16k buffer but could only call up the directory requester once. I was not able to open any more than two buffers at any time. If there has been a fix to this, MSS has not attempted to contact me and I'm a registered owner.

A new role for the Amiga?

I'd like to finish up with a speculation.

Current Amiga owners have a high degree of computer literacy. Despite efforts to make it a friendly machine, its power and its price have placed it out of reach of the home computerist. Whether that will change with the introduction of the 500 is debatable, although I hope and believe it will. Where the Amiga *will* find a niche, is in the narrowing gap between high-end personal computers and low-end engineering and graphic workstations. The Sun systems, for example, were originally designed and marketed as technical workstations. Yet, some 300 third-party business and desktop publishing programs have been developed for it. Meanwhile, the Mackintosh II has obviously been aimed at the workstation market and IBM's Personal System 2 will eventually have a windowing interface on a multitasking OS.

It's becoming apparent that users want high resolution, colour graphics for their personal computer in addition to speed and multitasking. The Amiga places this kind of power in the hands of humble individuals like me and you. It has the multitasking and windowing interface of the workstation and the low cost, mass market appeal of the personal computer.

I think that the most important development for the Amiga, then, will be in the areas previously dominated by much more expensive machines: computer-aided design, computer-aided engineering, artificial intelligence, audio-visual control, animation and publishing.

Try to remember that prophecy for the next five years and check my prognostication.

Any comments? Send them to me c/o The Transactor, on CompuServe 71426,1646 or PeopleLink AMTAG.

Mr. Ed

A Modular Text Editor For The Commodore 64

Chris Miller
Kitchener, Ontario

About Mr. Ed

Mr. Ed is an ASCII source editor/word processor. With it you will be able to create and modify large bodies of text effortlessly. The format of this text is very simple: only a carriage return, ie. CHR$(13), separates one line from the next. There is no tokenization, and no line numbering or link addressing as is the case with source written on the Basic editor. There is also no wasted or filler space as is the case with many word processors.

Assembly language source written on Mr. Ed will require less memory than if it had been written in Basic style. Converting an assembler like SYMASS to handle this source would be an easy thing to do indeed and probably reduce the complexity, while increasing the speed and reliability, of the program.

Mr. Ed is designed to provide a springboard for individuals who would like to create their own custom, top notch text editing environment without all the grind–work and nit–picking and preparation involved in laying a foundation.

What You Get

Although Mr. Ed is just less than 1K of code it is in some respects already a fairly sophisticated little package. Full bi–directional, four–way scrolling and paging is supported. Lines may be up to 250 characters long. The effect is like typing on a very long, wide document. Your screen "window" to this document moves side to side or up and down as you require.

The Status Line

The current line and column of the cursor and the number of free bytes of memory remaining are constantly updated on the status line at the top of the screen. The number of lines available is not fixed as with most word processors but depends on their average length; a blank line requires only one byte of memory. You can type right out to column 250. Just under 39,000 bytes will normally be available as work space.

Loading and Saving Text

Text may be loaded and saved via the Basic LOAD/SAVE "FILENAME",8 commands. Pressing RUN/STOP while in the editor will automatically return you to Basic where these or any immediate commands may be given. The Basic command "ED"<RETURN> will recall Mr. Ed with old source or newly LOAD'ed source in memory.

Editing Commands

1. The CRSR keys work pretty much the way you would expect. Use them to move (the cursor) or scroll (the window) up, down, left and right. You cannot CRSR out of the range of entered text.

2. The INST/DEL key is used to insert and delete one character at a time just the way it is in Basic.

3. The RETURN key is non–destructive and does nothing but advance you to the next line.

4. F1 deletes whole lines to the right and can be used to join two lines as well.

5. F2 inserts a line, ie. a CHR$(13). It can be used to split a line in two as well as open up space.

6. F3 and F4 allow vertical page scrolling. F3 takes you down and F4 up one screen at a time.

7. F5 and F6 control horizontal paging.F5 moves you one screen to the right and F6 one screen to the left.

8. RUN/STOP exits to Basic where any immediate commands may be used. ED<RETURN> will put you back in the editor.

Going up or down to a new line always places you at the beginning of it, ie. full window left.

If you would rather have other keys do these things simply replace their character values in the command list at the end of the code with ones of your choosing.

Tricks With Memory

Mr. Ed uses Basic pointers to define its own work space. Therefore you should always use the Basic NEW command before calling Mr. Ed for the first time, or to clear old text. Utilities which lower the top of Basic (55–56) will also be safe from Mr. Ed. You may create space for Basic programs by raising the bottom of Basic (43–44) before invoking Mr. Ed. The last byte of text entered will be pointed to by Basic's SOV pointer (45–46) during and following any work session.

Check It Out

```
LOAD"MR.ED",8,1 ;to put the machine code in memory
NEW             ; just to clear some of Basic's pointers
SYS52000        ; runs the program
```

Use Mr. Ed to write a letter to your Grandma. Mr. Ed is a What–You–See–Is–What–You–Get word processor. As you type past column 40 the screen window scrolls along with you. Don't type past column 80 if your printer can't handle it.

Printing Text

When you're all done press Run/Stop to return to Basic. Next,

```
SAVE "LETTER",8
```

Now run the following short Basic program with your printer on. Don't bother typing the REM statements.

```
LM  100 pg=55: rem lines per page
JC  110 open 4,4,7
JB  120 open 5,8,5,"letter"
LB  130 get#5,a$
AO  140 x=st: rem save status
ME  150 print#4,a$;
CN  160 if a$=chr$(13) then lc=lc+1
MC  170 if lc<>pg then 200
BF  180 get k$: if k$="" then 180
KG  190 lc=0: rem initialize line count
OC  200 if x<>64 then 130
BP  210 print#4: close4: close5
```

Your letter will be printed exactly the way it appeared on screen. Tomorrow you can LOAD "LETTER",8 back in and modify it a bit for your other Granny.

The above program could just as easily be written in assembler by anyone familiar with Kernal Rom. It would run quite a bit faster and could even be added as a command to Mr. Ed.

Season To Taste

Those who copy or otherwise acquire the source for Mr.Ed will be able to modify screen and text colours, screen window size and position, paging distances, maximum line size, and much more just by changing a few constants.

Create Your Own Recipes

The beauty of Mr. Ed lies in its highly structured design, and in its compactness and compatibility with Basic. New commands can be added to the source almost effortlessly. To introduce a new feature simply tack on, to the command list at the end of the source, the character value of any key press followed by the address–1 of the routine you would like executed when this key is pressed. A normal RTS from your routine will return control to Mr.Ed's main key scan loop.

Let The Fun Begin

My favourite phase of program development comes when all the fuss work is done and further coding involves primarily macro–like calls to existing routines. A little programming can go a very long way at this point and debugging is greatly simplified (by using stuff that already works). Mr. Ed is in just this stage of development. Although it may be considered by some to be a beautiful and complete, albeit simple, little text editor, I like to think of it as a pre–fabricated super–duper editor.

Some Staple Routines

Mr. Ed's WINDOW routine is the real workhorse and is used to move ASCII text from the buffer to the screen window. The TOP pointer will be set to point to the start of a text line. This line will appear at the top of the screen. TOP determines the vertical position of the window. The SHIFT variable determines the horizontal position of the window for left/right scrolling. DISFLG is used to enable/disable the window for operations where having display constantly updated may not be desirable (ie. speed is of the essence).

The LEFT and RIGHT routines allow lateral motion across any line; and UP and DOWN cause vertical travel through the text buffer, always to the start of a line. All display and pointer positioning overhead is taken care of.

FINDEOLN returns in .Y the number of characters remaining in the current line of text (as pointed to by TXT).

INSERT and DELETE are used to open up and remove space.

The MESSAGE routine can be used much the way Basic's PRINT command is. All ASCII text following JSR MESSAGE will be printed. This text must be followed by a zero byte; execution resumes on the byte following the zero.

There is no point in duplicating source comments here. If you decide to tinker with Mr.Ed, and I hope you will, you may want to check out the uses of its various pointers and variables for yourself.

A Project For ML Programmers

I would really love to have added SEARCH & REPLACE and CUT&PASTE commands, and even SPLITSCREEN and MULTI-BUFFER modes to Mr. Ed (and probably will someday), but I promised myself that I would keep this version short and sweet, and allow you to develop your own highly personalized editing tool.

Editor's Note Regarding The Source Code

Chris Miller is the author of the Buddy System–64 and Buddy System–128 assembler/editor package, available through Pro–Line Software. Buddy is a assembler that follows PAL format plus adds many more features for the ML programmer. In keeping this in mind, you will realize why Chris wrote Mr. Ed in Buddy format. For the magazine, though, we decided that we should stay with PAL format. On this issue's diskette we will include both the PAL and Buddy source files.

Mr. Ed: BASIC Loader

```
PD 1000 rem save "0:ed.ldr",8
CG 1010 rem by chris miller - kitchener, ontario
DL 1020 rem mr.ed - a text editor for the c64
KH 1030 :
MG 1040 for j = 52000 to 53022: read x: poke j,x
        : ch=ch+x: next
CC 1050 if ch<>134753 then print "checksum
        error!": stop
IA 1060 sys(52000): rem fire-up mr.ed
CK 1070 :
IF 1080 data 169, 128, 141, 138,   2, 169, 228, 141
JG 1090 data   4,   3, 169, 206, 141,   5,   3, 169
MG 1100 data   9, 141,  17, 208, 162,   6, 142,  32
LG 1110 data 208, 169,   1, 141,  33, 208,  32, 198
NP 1120 data 206,   5, 147,  14,   8, 158,   0, 142
FC 1130 data  33, 208, 169,  27, 141,  17, 208, 169
OL 1140 data   0, 162,   7, 149,   2, 202,  16, 251
FO 1150 data  32,  63, 206,  32, 187, 204, 169, 203
AA 1160 data  72, 169,  93,  72,  32,  75, 205,  32
BN 1170 data 121, 206,  32, 228, 255, 240, 251,  32
IH 1180 data  75, 205, 217, 248, 206, 240,   9, 200
FI 1190 data 200, 200, 192,  39, 144, 244, 176,   9
LD 1200 data 185, 250, 206,  72, 185, 249, 206,  72
MJ 1210 data  96, 170, 201,  13, 240,  12, 165,   4
GJ 1220 data 201, 210, 208,   6, 165,   3, 201,  39
KP 1230 data 240,  64,  32,  86, 205, 144,   5,  32
BG 1240 data 136, 205, 176,  54, 224,  13, 240,  84
MJ 1250 data 160,   0, 177, 253, 201,  13, 208,   3
OA 1260 data  32,  99, 205, 138, 145, 253,  32,  62
PO 1270 data 205, 145, 251,  32,  43, 206, 152, 240
EJ 1280 data  25, 230, 253, 208,   2, 230, 254, 165
KC 1290 data   3, 201,  39, 208,   5, 230,   4,  76
EA 1300 data 187, 204, 230,   3, 230, 251, 208,   2
MK 1310 data 230, 252,  96, 198,   4,  76, 187, 204
DP 1320 data 165,   3,   5,   4, 240,  21,  32,  51
LF 1330 data 205, 165,   3, 240, 238, 198,   3, 198
NG 1340 data 251, 165, 251, 201, 255, 208,   2, 198
FO 1350 data 252, 133, 251,  96,  32,  43, 206, 138
PF 1360 data 145, 253,  32,  43, 206, 168, 240, 243
CF 1370 data  32, 159, 204,  32,  43, 206,  32,  40
CH 1380 data 205, 230,   5, 208,   2, 230,   6, 165
IG 1390 data   2, 201,  23, 208,   3,  76, 146, 204
IG 1400 data 230,   2, 165, 251,  24, 105,  40, 144
MC 1410 data   2, 230, 252, 133, 251,  96, 165,   5
KM 1420 data   5,   6, 240,  67, 198,   5, 165,   5
ML 1430 data 201, 255, 208,   2, 198,   6, 165, 253
NP 1440 data  56, 229,   4, 176,   2, 198, 254, 133
IM 1450 data 253,  32, 159, 204, 160,   1, 132,   8
NK 1460 data 136,  32,  51, 205, 177, 253, 240,   8
EC 1470 data 201,  13, 208, 245, 198,   8,  16, 241
HL 1480 data 230, 253, 208,   2, 230, 254, 165,   2
IB 1490 data 240,  14, 198,   2, 165, 251,  56, 233
BB 1500 data  40, 176,   2, 198, 252, 133, 251,  96
IG 1510 data 160, 255, 198,  64, 136, 177,  63, 240
DO 1520 data   4, 201,  13, 208, 247,  56, 152, 101
DF 1530 data  63, 144,   2, 230,  64, 133,  63,  76
OA 1540 data 187, 204, 160, 255, 200, 177,  63, 240
DC 1550 data 236, 201,  13, 208, 247, 240, 230, 165
LG 1560 data 251,  56, 229,   3, 176,   2, 198, 252
AP 1570 data 133, 251, 165, 253,  56, 229,   3, 176
CD 1580 data   2, 198, 254, 133, 253, 169,   0, 133
MG 1590 data   3, 133,   4,  36,   9,  48,  83, 169
JI 1600 data  24, 133,   7, 165, 254,  72, 165, 253
GO 1610 data  72, 165, 252,  72, 165, 251,  72, 165
NP 1620 data  63, 133, 253, 165,  64, 133, 254,  32
LF 1630 data 112, 206, 160, 255,  32,  51, 206, 240
EK 1640 data  50, 196,   4, 144, 247,  24,  32,  41
FN 1650 data 205, 160, 255,  32,  51, 206, 240,  42
LN 1660 data  32,  62, 205, 145, 251, 192,  39, 144
LF 1670 data 242,  32,  45, 206,  32,  40, 205,  32
AD 1680 data  34, 204, 198,   7, 208, 212, 104, 133
FJ 1690 data 251, 104, 133, 252, 104, 133, 253, 104
BI 1700 data 133, 254,  96,  32,  40, 205, 160,   0
MJ 1710 data 240,   3,  32,  40, 205, 169,  32, 145
IP 1720 data 251, 200, 192,  40, 144, 249, 176, 215
DG 1730 data  56, 152, 101, 253, 144,   2, 230, 254
LC 1740 data 133, 253,  96, 198, 253, 165, 253, 201
FA 1750 data 255, 208,   2, 198, 254,  96,  73, 128
OJ 1760 data  16,   8,  73, 128, 201,  64, 144,   2
KC 1770 data  73,  64,  96,  72, 160,   0, 177, 251
CG 1780 data  73, 128, 145, 251, 104,  96, 165, 254
PE 1790 data 197,  46, 144,   6, 208,   4, 165, 253
CF 1800 data 197,  45,  96, 165,  46,  72, 165,  45
NA 1810 data  72, 160,   0, 177,  45, 200, 145,  45
HG 1820 data 136, 198,  45, 165,  45, 201, 255, 208
CF 1830 data   2, 198,  46,  32,  86, 205, 144, 235
BG 1840 data 240, 233, 104, 133,  45, 104, 133,  46
EI 1850 data 165,  46, 197,  56, 144,   6, 165,  45
CK 1860 data 197,  55, 176,   6, 230,  45, 208,   2
MP 1870 data 230,  46,  96, 169,   1, 133,   8,  32
DL 1880 data 224, 203, 165, 254,  72, 165, 253,  72
CI 1890 data 164,   8, 177, 253, 160,   0, 145, 253
OG 1900 data 230, 253, 208,   2, 230, 254,  32,  86
CI 1910 data 205, 144, 237, 240, 235, 104, 133, 253
JP 1920 data 104, 133, 254,  32,  86, 205, 176,  11
GM 1930 data 165,  45,  56, 229,   8, 133,  45, 176
HP 1940 data   2, 198,  46,  76, 187, 204,  32,  99
FN 1950 data 205, 169,  13, 208,   5,  32,  99, 205
LO 1960 data 169,  32, 145, 253,  76, 187, 204, 102
JK 1970 data   9,  32,  43, 206, 152, 208,   1, 200
BM 1980 data 132,   8,  32, 162, 205,  76,  38, 206
FK 1990 data 102,   9, 162,  23,  32,   2, 204, 202
KO 2000 data 208, 250, 240,  34, 102,   9, 162,  23
DB 2010 data  32,  46, 204, 202, 208, 250, 240,  22
HN 2020 data 102,   9, 162,  39,  32, 224, 203, 202
MP 2030 data 208, 250, 240,  10, 102,   9, 162,  39
HE 2040 data  32, 187, 203, 202, 208, 250,  70,   9
BN 2050 data  76, 187, 204, 160, 255,  32,  51, 206
GN 2060 data 208, 251,  96, 200, 192, 255, 240,   6
FE 2070 data 177, 253, 240,   2, 201,  13,  96, 165
GM 2080 data  43, 166,  44,  24, 105,   2, 144,   1
AE 2090 data 232, 133, 253, 133,  63, 134, 254, 134
HG 2100 data  64, 165,  45, 133,  61, 165,  46, 133
GC 2110 data  62, 160,   0, 152, 145,  61, 230,  61
BC 2120 data 208,   2, 230,  62, 166,  62, 228,  56
AK 2130 data 144, 242, 166,  61, 228,  55, 144, 236
OA 2140 data 169,  40, 133, 251, 169,   4, 133, 252
CA 2150 data  96,  32, 198, 206,  19, 195, 207, 204
BK 2160 data 213, 205, 206,  58,   0, 165,   4,  56
GF 2170 data 101,   3, 170, 152,  32, 205, 189,  32
```

```
FP 2180 data 198, 206,  32, 204, 201, 206, 197,  58
MI 2190 data   0, 164,   6, 166,   5, 232, 208,   1
DO 2200 data 200, 152,  32, 205, 189,  32, 198, 206
FE 2210 data  32, 198, 210, 197, 197,  58,   0, 165
EP 2220 data  56,  56, 229,  46, 168, 165,  55, 229
BF 2230 data  45, 170, 152,  32, 205, 189,  32, 198
LL 2240 data 206,  32,  32,  32,   0,  96, 160,   0
OC 2250 data 104, 133,  61, 104, 133,  62, 230,  61
HN 2260 data 208,   2, 230,  62, 177,  61, 240,   5
IJ 2270 data  32, 210, 255, 208, 241, 165,  62,  72
IG 2280 data 165,  61,  72,  96, 173,   0,   2, 201
OH 2290 data  69, 208,  10, 173,   1,   2, 201,  68
NA 2300 data 208,   3,  76,  47, 203,  76, 124, 165
LO 2310 data 148, 220, 205,  20, 154, 205, 133, 230
JA 2320 data 205, 137, 213, 205, 134, 247, 205, 138
JA 2330 data   3, 206, 135,  27, 206, 139,  15, 206
BE 2340 data   3, 122, 227,  17,   1, 204, 145,  45
KE 2350 data 204, 157, 223, 203,  29, 186, 203
```

Mr. Ed PAL Source Listing

```
LC 1000 rem save"0:ed.pal",8
OG 1010 open 8,8,1,"0:mr.ed"
NM 1020 sys700
JC 1030 .opt o8
MB 1040 ; "Mr. Ed by Chris Miller Jul, 1986"
AJ 1050 ;
EJ 1060 ; ••• constants •••
BF 1070 columns   =  40          ; screen size
PH 1080 linesize  =  250         ; max allowed
IE 1090 screenbeg =  1024+40     ; top of text scr
IL 1100 screenend =  2024        ; end of text scr
CC 1110 rows      =  24          ; screenend-screenbeg/columns
GN 1120 ;
EL 1130 ; ••• important memory •••
AC 1140 vic       =  $d011
GB 1150 bkg       =  53281
FC 1160 bor       =  53280
OE 1170 rptkey    =  650
HM 1180 icrunch   =  $304
IH 1190 input     =  $200
GC 1200 ;
MJ 1210 ; ••• rom routines •••
OH 1220 crunchsrv =  $a57c
EL 1230 getin     =  $ffe4
GB 1240 print     =  $ffd2
IC 1250 ready     =  $e37b
PD 1260 cnvrtdec  =  $bdcd
MG 1270 ;
JJ 1280 ; ••• variables •••
AE 1290 *         =  2
GI 1300 vars      =  *
GL 1310 row       *= *+1         ; screen row 0-24
HF 1320 col       *= *+1         ; screen col 0-39
JP 1330 shift     *= *+1         ; off screen left
GF 1340 line      *= *+2         ; text line counter
BF 1350 cnt       *= *+1         ; display line counter
DL 1360 num       *= *+1         ; general purpose
NM 1370 disflg    *= *+1         ; negative=no display
CM 1380 varnum    =  *-vars
EO 1390 ;
FE 1400 ; ••• pointers •••
HB 1410 ptr       =  61          ; utility pointer
KP 1420 top       =  63          ; top line of text window
AH 1430 sob       =  43          ; start of basic
DG 1440 eob       =  55          ; end of basic memory
IG 1450 end       =  45          ; end of text
FI 1460 txt       =  253         ; current text position
IL 1470 scr       =  251         ; current screen position
OD 1480 ;
EL 1490 ; ••• beginning of code •••
AA 1500 *         =  52000
GB 1510           lda  #128      ; keys repeat
HK 1520           sta  rptkey
AH 1530 ;
NG 1540           lda  #<crunchwdg ; wedge for basic
FH 1550           sta  icrunch
FC 1560           lda  #>crunchwdg
AH 1570           sta  icrunch+1
CK 1580 ;
FE 1590 ; entry for ed command from basic
GK 1600 start     =    *
HE 1610           lda  #9
GK 1620           sta  vic       ; screen off
EN 1630 ;
NJ 1640           ldx  #6        ; blue screen
GB 1650           stx  bor
JP 1660           lda  #1        ; white print
JN 1670           sta  bkg
IF 1680           jsr  message   ; y=0
BN 1690 .byte 5, 147, 14, 8, 158, 0
DF 1700           stx  bkg
EC 1710 ;
PK 1720           lda  #27       ; screen on
JB 1730           sta  vic
CE 1740 ;
DC 1750 topoftext =    *
LM 1760           lda  #0
ML 1770           ldx  #varnum-1 ; init vars
KG 1780 ;
JB 1790 b1        sta  vars,x
PL 1800           dex
KF 1810           bpl  b1
CJ 1820 ;
CM 1830           jsr  initialize
FO 1840           jsr  window
AL 1850 ;
KK 1860 ; main key scan loop
FJ 1870 getkey    =    *         ; always return here
GM 1880           lda  #>getkey-1: pha
EN 1890           lda  #<getkey-1: pha
CO 1900 ;
EA 1910           jsr  reverse
GL 1920           jsr  statusline ; line, col, mem
AA 1930 ;
CC 1940 b2        jsr  getin
LN 1950           beq  b2
OB 1960 ;
JH 1970           jsr  reverse   ; y=0
CD 1980 ;
FJ 1990 ; check command keys
JA 2000 b3        cmp  commands,y
CI 2010           beq  foundkey  ; also sets carry
IP 2020           iny: iny: iny
IM 2030           cpy  #commandnum
JP 2040           bcc  b3
OI 2050           bcs  put       ; a typing key
CI 2060 ;
DO 2070 foundkey  =    *         ; jump to routine
FD 2080           lda  commands+2,y: pha
LD 2090           lda  commands+1,y: pha
AC 2100           rts
EL 2110 ;
GJ 2120 ; put character in text buffer
LE 2130 put       =    *
IN 2140           tax            ; save key
BD 2150           cmp  #13
PK 2160           beq  f1
AP 2170 ;
ME 2180 ; see if line full
KM 2190           lda  shift
ID 2200           cmp  #linesize-40
MM 2210           bne  f1
JN 2220           lda  col
LF 2230           cmp  #columns-1
LA 2240           beq  r1
AE 2250 ;
HA 2260 f1        jsr  testpos   ; are we at end
BO 2270           bcc  f2
OF 2280 ;
JO 2290           jsr  pshend    ; make room if can
PA 2300           bcs  r1        ; out of memory
MH 2310 ;
OJ 2320 f2        cpx  #13
HL 2330           beq  cret
KJ 2340 ;
JH 2350           ldy  #0
EF 2360           lda  (txt),y
CM 2370           cmp  #13       ; end of a line check
KH 2380           bne  f3
HB 2390           jsr  insert
PK 2400 f3        txa
EM 2410           sta  (txt),y
NF 2420           jsr  cnvscr
PI 2430           sta  (scr),y
OP 2440 ;
HG 2450 ; cursor right routine
OM 2460 right     =    *         ; cursor right routine
```

```
ME 2470            jsr  findeoln
HG 2480            tya
HH 2490            beq  r1          ; already at end
ME 2500            inc  txt
OP 2510            bne  f4
JD 2520            inc  txt+1
BF 2530 f4         lda  col
BJ 2540            cmp  #columns-1
IC 2550            bne  f5
GH 2560 ;
GG 2570            inc  shift
PK 2580            jmp  window
EJ 2590 ;
MJ 2600 f5         inc  col
JI 2610            inc  scr
CH 2620            bne  r1
GH 2630            inc  scr+1
JM 2640 r1         rts
AN 2650 ;
FD 2660 ; cursor left routine
FN 2670 b4         dec  shift       ; scroll left
DB 2680            jmp  window
IP 2690 ;
HM 2700 left       =    *
DM 2710            lda  col
GF 2720            ora  shift       ; check position
BL 2730            beq  r2          ; cant go left
AH 2740            jsr  dectxt
LO 2750            lda  col
JA 2760            beq  b4
KP 2770            dec  col
OA 2780            dec  scr
NB 2790            lda  scr
CP 2800            cmp  #$ff
OC 2810            bne  f6
PA 2820            dec  scr+1
FL 2830 f6         sta  scr
EJ 2840 r2         rts
IJ 2850 ;
JE 2860 ; carriage return handling routine
BF 2870 cret       =    *           ; handle carriage return
GO 2880            jsr  findeoln
GP 2890            txa              ; x=13
OK 2900            sta  (txt),y
EN 2910 ;
DD 2920 ; cursor down routine
JP 2930 down       =    *
CC 2940            jsr  findeoln
FF 2950            tay
NN 2960            beq  r2          ; allready at bottom
DH 2970            jsr  unshift     ; all the way left
KE 2980            jsr  findeoln
MH 2990            jsr  addy
MB 3000            inc  line
IP 3010            bne  f7
MD 3020            inc  line+1
GI 3030 f7         lda  row
EK 3040            cmp  #rows-1     ; last row check
CC 3050            bne  f8
KF 3060            jmp  topdown     ; scroll down
DL 3070 f8         inc  row
OH 3080 ;
JN 3090 addrow     =    *
DF 3100            lda  scr
MJ 3110            clc
FH 3120            adc  #columns
LE 3130            bcc  f9
EH 3140            inc  scr+1
OP 3150 f9         sta  scr
EE 3160            rts
IN 3170 ;
EH 3180 ; cursor up routine
EI 3190 up         =    *
CD 3200            lda  line        ; check position
JG 3210            ora  line+1      ; check position
NI 3220            beq  r3          ; at top already
NN 3230            dec  line
MO 3240            lda  line
EL 3250            cmp  #$ff
GH 3260            bne  f10
BB 3270            dec  line+1
GJ 3280 f10        lda  txt
LF 3290            sec
AD 3300            sbc  shift
CM 3310            bcs  f11
ED 3320            dec  txt+1
DP 3330 f11        sta  txt
KP 3340            jsr  unshift     ; scroll far left
DG 3350            ldy  #1
FA 3360            sty  num
FO 3370            dey
BA 3380 b5         jsr  dectxt      ; go back 2 cr
KF 3390            lda  (txt),y
NB 3400            beq  f12
NB 3410            cmp  #13
KI 3420            bne  b5
IK 3430            dec  num
AM 3440            bpl  b5
AP 3450 ;
EG 3460 f12        inc  txt         ; then forward 1 char
BF 3470            bne  f13
JP 3480            inc  txt+1
JB 3490 f13        lda  row         ; top of screen check
HC 3500            beq  topup       ; scroll up
MC 3510 ;
NG 3520            dec  row         ; else move up row
BA 3530            lda  scr
FF 3540            sec
BE 3550            sbc  #columns
FM 3560            bcs  f14
NP 3570            dec  scr+1
MN 3580 f14        sta  scr
FI 3590 r3         rts
GI 3600 ;
AN 3610 ; move window up
GI 3620 topup      =    *           ; move text window up line
LE 3630            ldy  #$ff
BK 3640            dec  top+1
PG 3650 b6         dey
GJ 3660            lda  (top),y
HM 3670            beq  newtop
LC 3680            cmp  #13
KJ 3690            bne  b6
KO 3700 ;
JL 3710 newtop     =    *           ; add y to top pointer
JA 3720            sec
JE 3730            tya
KD 3740            adc  top
GE 3750            bcc  f15
OD 3760            inc  top+1
LE 3770 f15        sta  top
PF 3780            jmp  window
EE 3790 ;
IL 3800 ; move window down
BE 3810 topdown    =    *           ; move window down line
JA 3820            ldy  #$ff
DE 3830 b7         iny
KE 3840            lda  (top),y
LH 3850            beq  newtop
PN 3860            cmp  #13
AF 3870            bne  b7
JJ 3880            beq  newtop
IK 3890 ;
NE 3900 ; initialize for start of new line
EP 3910 unshift    =    *
HI 3920            lda  scr
LN 3930            sec
BK 3940            sbc  col
BF 3950            bcs  f16
DI 3960            dec  scr+1
KG 3970 f16        sta  scr
CA 3980 ;
OP 3990            lda  txt
BC 4000            sec
HO 4010            sbc  col
KJ 4020            bcs  f17
KP 4030            dec  txt+1
BN 4040 f17        sta  txt
IE 4050 ;
HM 4060            lda  #0
BF 4070            sta  col
KG 4080            sta  shift
AH 4090 ;
FI 4100 ; move text to screen window
KC 4110 window     =    *
PP 4120            bit  disflg      ; is display on
DK 4130            bmi  r4          ; no
CK 4140 ;
NG 4150            lda  #rows       ; screenend-screenbeg
                                    /columns
BM 4160            sta  cnt
AM 4170 ;
PO 4180            lda  txt+1: pha  ; save pointers
BF 4190            lda  txt: pha
ND 4200            lda  scr+1: pha
ED 4210            lda  scr: pha
CP 4220 ;
JN 4230            lda  top: sta txt    ; init pointers
HL 4240            lda  top+1: sta txt+1
OE 4250            jsr  initscr
KB 4260 ;
OL 4270 ; process next line of text
CK 4280 newline    =    *
PN 4290            ldy  #$ff
FB 4300 b8         jsr  testeoln
JG 4310            beq  lineblank
PG 4320            cpy  shift       ; handle right scroll
FP 4330            bcc  b8
KG 4340 ;
EH 4350            clc
FA 4360            jsr  addy+1      ; update txt ptr
II 4370 ;
JD 4380            ldy  #$ff
CH 4390 b9         jsr  testeoln
EH 4400            beq  restblank   ; end of line
EN 4410            jsr  cnvscr      ; convert ascii code
PD 4420            sta  (scr),y     ; put on screen
AC 4430            cpy  #columns-1
FG 4440            bcc  b9
IN 4450 ;
MA 4460            jsr  findeoln+2  ; dont init .y
CF 4470            jsr  addy        ; point next line
GP 4480 ;
LC 4490 addscr     =    *
KN 4500            jsr  addrow      ; to screen ptr
MN 4510            dec  cnt
ML 4520            bne  newline
IC 4530 ;
MA 4540 fini       =    *
MK 4550            pla: sta scr     ; restore ptrs
AO 4560            pla: sta scr+1
CE 4570            pla: sta txt
FC 4580            pla: sta txt+1
AH 4590 r4         rts
OG 4600 ;
IA 4610 lineblank  =    *
KN 4620            jsr  addy
BG 4630            ldy  #0
HA 4640            beq  f18
DF 4650 restblank  =    *
CA 4660            jsr  addy
GM 4670 f18        lda  #' '
OM 4680 b10        sta  (scr),y
CD 4690            iny
BC 4700            cpy  #columns
DP 4710            bcc  b10
MF 4720            bcs  addscr      ; start next line
AP 4730 ;
LD 4740 addy       =    *           ; add y to text ptr
PA 4750            sec
PE 4760            tya
MP 4770            adc  txt
IF 4780            bcc  f19
HB 4790            inc  txt+1
BN 4800 f19        sta  txt
GL 4810            rts
KE 4820 ;
BD 4830 dectxt     =    *           ; back up text ptr
LE 4840            dec  txt
KF 4850            lda  txt
OP 4860            cmp  #$ff
CM 4870            bne  f20
ME 4880            dec  txt+1
PO 4890 f20        rts
KJ 4900 ;
BD 4910 ; convert ascii to screen code
DE 4920 cnvscr     =    *
FK 4930            eor  #128
NC 4940            bpl  f21
JL 4950            eor  #128
ID 4960            cmp  #64
AA 4970            bcc  f21
ID 4980            eor  #64
HF 4990 f21        rts
OP 5000 ;
MC 5010 reverse    =    *           ; reverse cursor char
IB 5020            pha
BP 5030            ldy  #0
DI 5040            lda  (scr),y
EA 5050            eor  #$80
FN 5060            sta  (scr),y
GF 5070            pla
EM 5080            rts
IF 5090 ;
```

```
ON 5100 testpos   =   *          ; test position in text
HD 5110           lda  txt+1
EB 5120           cmp  end+1
DK 5130           bcc  f22
GN 5140           bne  f22
GI 5150           lda  txt
FF 5160           cmp  end
PA 5170 f22       rts
CL 5180 ;
NP 5190 insert    =    *           ; insert one space
KK 5200           lda  end+1: pha
HA 5210           lda  end: pha
PK 5220           ldy  #0
KD 5230 b11       lda  (end),y
IF 5240           iny
EN 5250           sta  (end),y
HE 5260           dey
JH 5270           dec  end
II 5280           lda  end
MK 5290           cmp  #$ff
JH 5300           bne  f23
II 5310           dec  end+1
HL 5320 f23       jsr  testpos
CG 5330           bcc  b11
KK 5340           beq  b11
OM 5350           pla: sta end
PL 5360           pla: sta end+1
AH 5370 ;
NP 5380 pshend    =    *           ; bump end ptr up
NN 5390           lda  end+1
HE 5400           cmp  eob+1
BM 5410           bcc  f24
EB 5420           lda  end
MI 5430           cmp  eob
FJ 5440           bcs  r5
KO 5450 f24       inc  end
CJ 5460           bne  r5
NE 5470           inc  end+1
NO 5480 r5        rts
IO 5490 ;
IO 5500 deletechr =    *
DH 5510           lda  #1
FB 5520           sta  num
GE 5530           jsr  left        ; dim
NF 5540 delete    =    * ; number of chars in num
EL 5550           lda  txt+1: pha
LK 5560           lda  txt: pha
HK 5570 b12       ldy  num
IO 5580           lda  (txt),y
BC 5590           ldy  #0
KD 5600           sta  (txt),y
CH 5610           inc  txt
PL 5620           bne  f25
PF 5630           inc  txt+1
PP 5640 f25       jsr  testpos
FK 5650           bcc  b12
NO 5660           beq  b12
OI 5670           pla: sta txt
BH 5680           pla: sta txt+1
PO 5690           jsr  testpos
JC 5700           bcs  f26
GD 5710           lda  end
JN 5720           sec
JL 5730           sbc  num
CJ 5740           sta  end
LF 5750           bcs  f26
KE 5760           dec  end+1
IP 5770 f26       jmp  window
KA 5780 ;
AB 5790 insertln  =    *           ; insert chr$(13)
JG 5800           jsr  insert
ID 5810           lda  #13
NI 5820           bne  f27
MD 5830 ;
CA 5840 insrtspc  =    *           ; insert blank
LJ 5850           jsr  insert
JB 5860           lda  #" "
MM 5870 f27       sta  (txt),y
DJ 5880           jmp  window
IH 5890 ;
GA 5900 deleteln  =    *           ; delete line
ED 5910           ror  disflg      ; display off
GM 5920           jsr  findeoln
BO 5930           tya
IA 5940           bne  f28
OB 5950           iny
HH 5960 f28       sty  num
BK 5970           jsr  delete
AJ 5980           jmp  setwindow
MN 5990 ;
FK 6000 pagedown  =    *           ; long scroll down
KP 6010           ror  disflg      ; no display
BP 6020           ldx  #rows-1     ; 23 lines
OE 6030 b13       jsr  down
PE 6040           dex
BG 6050           bne  b13
ML 6060           beq  setwindow
MC 6070 ;
NJ 6080 pageup    =    *           ; long scroll up
KE 6090           ror  disflg      ; no display
DE 6100           ldx  #rows-1     ; 24 lines
PE 6110 b14       jsr  up
PJ 6120           dex
EL 6130           bne  b14
MA 6140           beq  setwindow
MH 6150 ;
PA 6160 pageleft  =    *
GB 6170           ror  disflg
AO 6180           ldx  #columns-1
MP 6190 b15       jsr  left
PO 6200           dex
HA 6210           bne  b15
MF 6220           beq  setwindow
MM 6230 ;
JF 6240 ; scroll sideways to end of line
PH 6250 pageright =    *
EP 6260           ror  disflg      ; no display
KD 6270           ldx  #columns-1
EK 6280 b16       jsr  right
JE 6290           dex
EG 6300           bne  b16
BB 6310 setwindow =    *
KE 6320           lsr  disflg
IB 6330           jmp  window      ; display
KD 6340 ;
MK 6350 ; set y=distance to text eol
IB 6360 findeoln  =    *
PP 6370           ldy  #$ff
BO 6380 b17       jsr  testeoln
BM 6390           bne  b17
MO 6400           rts
AI 6410 ;
II 6420 testeoln  =    *
OP 6430           iny
HF 6440           cpy  #$ff
OB 6450           beq  f29
IF 6460           lda  (txt),y
CD 6470           beq  f29
LB 6480           cmp  #13
DF 6490 f29       rts
KN 6500 ;
HC 6510 initialize =   *
HJ 6520           lda  sob
GN 6530           ldx  sob+1
CA 6540           clc
HH 6550           adc  #2
FD 6560           bcc  f30
GI 6570           inx
HK 6580 f30       sta  txt
IK 6590           sta  top
DK 6600           stx  txt+1
KN 6610           stx  top+1
GK 6620           lda  end: sta ptr
FF 6630           lda  end+1: sta ptr+1
DF 6640           ldy  #0          ; fill zeros
BL 6650           tya
JM 6660 b18       sta  (ptr),y
EI 6670           inc  ptr
JN 6680           bne  f31
BH 6690           inc  ptr+1
LO 6700 f31       ldx  ptr+1
OI 6710           cpx  eob+1
FO 6720           bcc  b18
MP 6730           ldx  ptr
DN 6740           cpx  eob
DA 6750           bcc  b18
NB 6760 initscr   =    *
FM 6770           lda  #<screenbeg: sta scr
NI 6780           lda  #>screenbeg: sta scr+1
CH 6790           rts
GA 6800 ;
PI 6810 statusline =   *
DD 6820           jsr  message
KH 6830 .byte 19                   ; (home)
EN 6840 .asc "COLUMN: ": .byte 0
OP 6850           lda  shift
NE 6860           sec
FP 6870           adc  col
LK 6880           tax
AN 6890           tya              ; =0
JK 6900           jsr  cnvrtdec
NI 6910           jsr  message
CD 6920 .asc "  LINE: ": .byte 0
CM 6930           ldy  line+1
ML 6940           ldx  line
CA 6950           inx
EP 6960           bne  f32
KB 6970           iny
KP 6980 f32       tya
DA 6990           jsr  cnvrtdec
HO 7000           jsr  message
NH 7010 .asc "  FREE: ": .byte 0
GF 7020           lda  eob+1
HP 7030           sec
PF 7040           sbc  end+1
JF 7050           tay
FK 7060           lda  eob
GJ 7070           sbc  end
DH 7080           tax
JG 7090           tya
BH 7100           jsr  cnvrtdec
FF 7110           jsr  message
HH 7120 .asc "   ": .byte 0        ; 3 (spaces)
GM 7130           rts
KF 7140 ;
FB 7150 ; print in source messages
FH 7160 message   =    *
NE 7170           ldy  #0
EJ 7180           pla
KK 7190           sta  ptr
IK 7200           pla
HJ 7210           sta  ptr+1
FB 7220 b19       inc  ptr
FA 7230           bne  f33
HJ 7240           inc  ptr+1
KO 7250 f33       lda  (ptr),y
LD 7260           beq  f34
EN 7270           jsr  print
BE 7280           bne  b19
AA 7290 f34       lda  ptr+1
AA 7300           pha
EO 7310           lda  ptr
EB 7320           pha
OI 7330           rts
CC 7340 ;
EI 7350 ; wedge for ed command in basic
KA 7360 crunchwdg =    *          ; wedge for ed command
HL 7370           lda  input
PD 7380           cmp  #"e"
LK 7390           bne  f35
ML 7400           lda  input+1
KF 7410           cmp  #"d"
JM 7420           bne  f35
FP 7430           jmp  start       ; call mr. ed
AK 7440 f35       jmp  crunchsrv   ; pass to basic
AJ 7450 ;
CK 7460 ; command entries
NO 7470 commands  =    *
NA 7480 .byte 148: .word insrtspc-1    ;inst
IK 7490 .byte  20: .word deletechr-1   ;del
JO 7500 .byte 133: .word deleteln-1    ;f1
CC 7510 .byte 137: .word insertln-1    ;f2
OM 7520 .byte 134: .word pagedown-1    ;f3
FA 7530 .byte 138: .word pageup-1      ;f4
JK 7540 .byte 135: .word pageright-1   ;f5
FO 7550 .byte 139: .word pageleft-1    ;f6
GG 7560 .byte   3: .word ready-1       ;run stop
GB 7570 .byte  17: .word down-1        ;(cursor down)
EO 7580 .byte 145: .word up-1          ;(cursor up)
GD 7590 .byte 157: .word left-1        ;(cursor left)
NJ 7600 .byte  29: .word right-1       ;(cursor right)
MP 7610 commandnum = *-commands
```

Mandelbrot Halo

Aubrey Stanley
Mississauga, Ontario

A Mandelbrot–set exploration program for the C128

The Mandelbrot set lies mysteriously at the centre of the two dimensional complex plane. Surrounding it is a halo whose splendor is discovered with the aid of a microscope powerful enough to penetrate the heart of the atom. When we magnify small areas, we unveil pictures of immense beauty. Repeatedly magnifying a single spot reveals scene upon scene of ever changing loveliness. Miniature replicas of the set appear out of nowhere, each a little different. There seemingly is no end to the process!

MANDELBROT HALO for the C128 will let you explore the halo in a manner that is both entertaining and instructive. A mathematical appreciation is not required, although readers so inclined will enjoy the article on the Mandelbrot set (Scientific American, Aug. 1985), by A. K. Dewdney, to whom I am deeply indebted for the inspiration. Coincidently, while developing the program, I came across Peter Schroeder's implementation for the Amiga (BYTE, Dec. 1986), but it in no way influenced this program.

The pictures take some time to complete. The initial picture of the Mandelbrot set with the halo takes 105 minutes. Times for magnified pictures generally take up to 60 minutes, while some extreme shots may take up to four hours. But if, like me, you crave to see the life that vibrates in every atom, your patience will not go unrewarded.

Typing It In

MANDELBROT HALO is in two parts. Program 1 ("halo.bas" on the disk) uses friendly C128 BASIC to control the user interface. Program 2 is in machine language and contains the floating point calculations and screen plotting routines which would run too slowly in BASIC. Incidentally, the original All–BASIC version took 20 hours to plot the Mandelbrot set!

To help you enter the program accurately, you should use the C128 "verifizer". See the section in the magazine on typing in programs.

Save the BASIC program under any name. The loader (Program 2) will create the machine language portion and name it "halo.obj"; this program is loaded by the BASIC program.

A Window On The Picture

Pictures are drawn in the graphics area of a split screen. The parent picture is the Mandelbrot set which must be plotted first and saved to disk. It may then be recalled whenever you want to select a new area of the halo for magnification. When the second picture is plotted, any portion of it may be selected for further magnification. This process may go on indefinitely.

The following description will help increase your enjoyment of the program. You may gloss over the math parts if you wish.

Certain parameters or values are associated with the area being magnified. These are displayed in the menu section of the split screen. The values are automatically manipulated as you move a Zoom Window over the picture. You may vary the size of the window and also the size of the graphics area upon which the contents of the window will be projected.

RE is the real part of the complex number representing the top left hand corner of the window. IM is the imaginary part of the same number. SI is the length of each side of the window. These three parameters define the area under the window.

In terms of pixels, the top left hand corner of the picture over which the window moves is given x:y coordinates of 0:0. The top left hand corner of the window is identified by X and Y, which are relative to coordinates 0:0. Z is the length in pixels of each side of the window and you may vary it from a single pixel to 24 pixels. X, Y, Z are related to RE, IM, SI respectively and are the more visible definitions of the area to be plotted. When encoded into a filename, they can help to trace the origin of a picture.

PIXELS determines the size of the projection area and is the length in pixels of each side. You may vary it from 16 to 160 pixels, so even a postage stamp size picture may be plotted. As smaller pictures will plot in less time, you can use this feature to quickly evaluate the result of a magnification, before deciding to go ahead with the maximum size. PIXELS corresponds to SI in terms of the visible screen area over which the window will be plotted, for example a 160 by 160 square pixel area.

So we calculate a value for gap by dividing SI by PIXELS. We now have a two dimensional array, 160 by 160 gaps, in terms of

the complex plane whose top left hand corner is RE/IM. For each gap in the vertical (imaginary) plane, we perform a repetitive operation on each gap in the horizontal (real) plane, a total of 160 times 160, or 25600 operations. This iterative operation is given by the equation, $z = z^2+c$, where c is the complex number containing the real and imaginary parts of gap and z is initially set to the complex number 0. Each time we calculate z, we substitute its value into the equation and then recalculate z. Repeatedly computing z in this way produces the Mandelbrot set, which is the set of numbers for which the size of z remains finite no matter how many times we recalculate it. Other numbers will tend to infinity, some sooner than others.

Numbers outside the Mandelbrot set are identified when z reaches a size of 2 (or more). Those within the set never reach this size. So we begin an iterative loop with a count of 1, calculate z, and repeat the loop if the result is below 2, stepping the count each time. If after 150 iterations, the size of z remains below 2, the pixel is assumed to lie in the set and is assigned a count of 0 for convenience. Pixels reaching a size of 2 in the process will exit the loop and retain the count at which they did so. Therefore pixels with a count of 2 lie within the set, those with small counts are very far from the set, and those with large counts are close to the set.

Colours are assigned to pixels according to how far from the set they lie. As only four colours may be used in C128 multicoloured mode without losing detail, black is assigned to pixels within the set and three other colours are distributed in spectrums based on a modulus–3 derivation of each pixel count. Each of these three colours may be individually changed.

For speed reasons, no scaling is employed in the plots. Therefore pictures are more rectangular than square, but this does not detract from their beauty. Remember that in terms of the complex plane, the pictures are truly square!

The Mandelbrot Set

Before you can begin to explore the halo, you must first generate the Mandelbrot set.

Load and run the BASIC program. You will see the menu in the text portion of the split screen, a colour line above it, and an all–black graphics screen. The PLOT option will be highlighted in the menu.

Press RETURN. The plot will commence at a brisk pace, then will slow down considerably in the region of the halo, near the set. You will not actually see the pixels within the set being plotted because they are left in the background colour (black).

The screen will go blank if you press down the CAPS LOCK key while the picture is being drawn. This is because the program goes into fast mode and the VIC chip cannot display at the (doubled) processor speed. Releasing the key will put the program into slow mode again and the picture will reappear. It makes sense to leave the CAPS key down because pictures will be drawn twice as fast, making for considerable savings in time. You can always release it from time to time in order to monitor progress.

Make sure to save the picture as soon as it is complete. To do so, use the Cursor Left key to move the highlight to the FILENAME option and then press RETURN. Release the CAPS key if you had left it down. Now type in a filename for the picture and press RETURN. Cursor Right to the SAVE option and press RETURN. Once the picture is saved to disk, you may recall it at any time by entering the filename as before, and pressing RETURN on the LOAD option.

To exit the program, move to the EXIT option and press RETURN. Sometimes you may wish to abort a picture while it is being drawn. In that case use the RUNSTOP/RESTORE combination. You may RUN the program again to start another picture.

Exploring The Halo

Bordering the Mandelbrot set is a halo of colour suffused with filaments that spread out in all directions. From our distant perspective its beauty is as yet undifferentiated, like the plumage of the peacock that lies at first in the plasma of its egg.

Two categories of pictures may be derived from the halo. The first contains exotic structures, often with miniature, imperfect versions of the set suspended like black jewels on filigreed tendrils. These pictures take a relatively short time to plot and occur when the window does not include any of the (black) area within the set proper. The second category occurs when we magnify tiny (one or two pixel) protrusions into the set. These often give spectacular, landscaped effects, but take a relatively long time to plot as some portion of the set must be included.

Start with the picture of the Mandelbrot set. Load it from disk if you need to, as explained in the last section. Move to the PLOT option and then press the SPACE bar. The action now moves to the graphics screen where a flashing window will appear in the centre of the picture. Its size may be varied by pressing the " + " and " – " keys. Move the window by using the four Cursor keys. The parameter displays are automatically updated to accommodate the size and position of the window.

Move around the halo looking for interesting spots. There are hundreds to choose from. If you plot outside the halo you will only get solid colour, while within the set there is only black! It is in the filamented border of the halo that you will find the variegated beauty you are looking for.

To move back to the menu, press the SPACE bar again. You may move back and forth in this manner. To vary the size of the projection area, move to the PIXELS option and use the " + " and " – " keys. To quickly evaluate the result of a magnification, select a small size, say 48 pixels. The picture will be plotted in a considerably less amount of time and may then be re-projected onto a larger area as long as you do not return to the graphics screen; otherwise the parameters will readjust to the window

position and size. It is always a safe bet to save the original picture first. Once you have selected the area for magnification and noted its X, Y and Z values (a good idea), press RETURN on the PLOT option. The current picture will be erased and the new one drawn. You may then save the picture and/or select a further area for magnification.

The Colour Options

Pixels in the Mandelbrot set are always plotted in black. The three other colours, however, may be changed individually for optimum presentation of each picture. Move to the (one of three) colour options – COL1, COL2 or COL3 – and use the " + " key to cycle through the colours.

The Colour Line above the menu shows the three colours currently in effect. It will change to reflect the colour changes you make. If a picture is displayed, it too will instantly be updated with the colour changes.

Use the " – " key to revert to the original colour of the last picture plotted or recalled to the screen.

Beyond The C128

The C128 does give very pleasing results in spite of its limitations: 160 by 160 resolution and only four colours. Where more colours are available, the results will improve dramatically. I have in mind here other graphics devices like colour printers and plotters, even other computers like the Amiga!

For this reason, MANDELBROT HALO will generate an array in memory by storing the count values reached for each pixel during the iterative loop mentioned earlier. To save the array on disk, enter the filename, then press RETURN on the DUMP option. If you want to view it, you will have to exit and use the Monitor.

As generating the array adds about 15 minutes to the picture, this function can be bypassed by running the program from the 40 column screen. The DUMP option is only in effect when you begin the program from the 80 column screen, then switch to the 40 column screen to use the program.

The array is stored as a program file, so if you are transferring it to another machine, remember that the first two bytes contain the start address. You can load the array into memory with the command BLOAD "filename",B0.

In C128 memory the array exists in bank 0, starting at address 6FFD hex. The format is as follows:

6FFD	LOWEST	Count in the array (excluding 0).
6FFE	HIGHEST	Count in the array.
6FFF	LENGTH	Of each line in the array, in pixels.
7000	COUNTS	For pixels in line 1, one byte each.
7000 + (LENGTH)	COUNTS	For each pixel in line 2.
7000 + (LENGTH * 2)	COUNTS	For each pixel in line 3.
etc.		

Remember that the array is square, so the number of lines is equal to the length of each line. The Low and High counts give the values for pixels outside the set, and can range from 1 to 151. A count of 0 in the array denotes a pixel lying in the set (black in the pictures).

Before plotting to an external device, it is a good idea to divide up the available colours only within the range of counts that exist in the array (excluding 0). That is why the Low/High counts are stored. You will use all the colours this way. An alternative is to simply assign the " n " colours to the Modulus–n values of each count. Pixels lying in the Mandelbrot set, (count = 0) must, of course, be assigned their own, unique colour.

Happy Exploring!

Mandelbrot Halo

```
LB  10 rem "mandelbrot halo" "aubrey stanley"
        "april 1987"
KJ  20 rem = the next line must be entered exactly
        as shown =
BN  30 re=-2:im=1.25:si=2.5:gp=si/160:cn=151
IN  35 r1$=chr$(18): r2$=chr$(146): cl$=chr$(157)
        : cr$=chr$(29)
NG  36 bl$=chr$(159): rd$=chr$(150): br$=chr$(149)
        : gy$=chr$(155)
FG  37 cu$=chr$(145): cd$=chr$(17)
DI  40 bank15:fast:close 15:open 15,8,15
        :bload "halo.obj"
EF  50 gosub 1770:gosub 930:close 15:end
NA  60 rem = = "print parameters" = =
NL  70 print gy$;:window 3,21,19,21,1:print ac;
JI  80 window 23,21,39,21,1:print bc;
EG  90 window 3,22,19,22,1:print cc;:return
CG  100 rem = = "print coords" = =
MD  110 if zm=0 then mx=0:my=0:mz=wd
        :else mz=sd
AA  120 print gy$;:window 21,22,26,22,1:print mx;
NC  130 window 28,22,33,22,1:print my;
MG  140 window 35,22,39,22,1:print mz;:return
KK  150 rem = = "print pixels" = =
EJ  160 print gy$;:window 33,23,38,23,1:print wd;
        :return
NG  170 rem = = "print filename" = =
FC  180 print gy$;:window 8,23,25,23,1:print f$;
KN  190 return
EN  200 rem = = "plot color line" = =
MO  210 gosub 220:gosub 230:goto 240
AG  220 box 1,0,164,52,167,,1:return
AM  230 box 2,53,164,105,167,,1:return
DC  240 box 3,106,164,159,167,,1:return
NO  250 rem = = "plot frame" = =
ED  260 color 3,12:for y=168 to 199:draw 3,0,y to 159,y
        :next:color 3,c3
CH  270 x=pa(px,1):y=wd-1:box 3,x,x,x+y,x+y:return
DA  280 rem = = "color values" = =
```

```
BP | 290 color 1,c1:color 2,c2:color 3,c3:graphic 3,1
   |     :return
IE | 300 color 1,c1:color 2,c2:if wi<>0 then sys 27456
   |     :else gosub 220:gosub 230
CF | 310 return
GM | 320 color 3,c3:if wi<>0 then sys 27415
IB | 330 gosub 240:return
AI | 340 c4=c1:c5=c2:c6=c3:return
DB | 350 rem == "plot values" ==
JA | 360 re=ac:im=bc:si=cc:sprite sa(sp,0),0
AL | 370 xc=pa(px,1):yc=xc+wd-1:gosub 70
   |     :gosub 110:gosub 160:return
CB | 380 rem == "make sprite" ==
GL | 390 sshape a$,0,0,23,20:sprsav a$,a:a=a+1
   |     :scnclr 1:return
FL | 400 rem == "print menu" ==
HN | 410 print chr$(19)chr$(19)rv$:scnclr 0:sys 51941
JO | 420 window 0,21,2,21:print br$"re";
MH | 430 window 20,21,22,21:print "im";
MN | 440 window 0,22,2,22:print "si";
BE | 450 window 20,22,35,22:print"x"spc(6)"y"
   |     spc(6)"z";
KD | 460 gosub 480:gosub 490:gosub 500:gosub 510
   |     :gosub 520
HF | 470 gosub 530:gosub 540:gosub 550:gosub 560
   |     :goto 570
HP | 480 window 0,23,8,23:print rv$;rd$;"filename";
   |     :goto 580
LA | 490 window 27,23,33,23:print rv$;rd$;"pixels";
   |     :goto 580
JH | 500 window 0,24,3,24:print rv$;bl$;"plot";
   |     :goto 580
DE | 510 window 5,24,8,24:print rv$;bl$;"load";
   |     :goto 580
PG | 520 window 10,24,13,24:print rv$;bl$;"save";
   |     :goto 580
CJ | 530 window 15,24,18,24:print rv$;bl$;"dump";
   |     :goto 580
HF | 540 window 20,24,23,24:print rv$;bl$;"col1";
   |     :goto 580
GH | 550 window 25,24,28,24:print rv$;bl$;"col2";
   |     :goto 580
DH | 560 window 30,24,33,24:print rv$;bl$;"col3";
   |     :goto 580
CI | 570 window 35,24,38,24:print rv$;bl$;"exit";
AD | 580 rv$=r2$:print rv$;:return
FA | 590 rem == "get key" ==
BK | 600 get k$:a=1
IH | 610 do until a=k
OG | 620 if k$=mid$(kb$,a,1) then exit:else a=a+1
OE | 630 loop
FI | 640 if a=k then 600:else k=a:return
LD | 650 rem == "load vars" ==
PB | 660 for a=0 to 4:bank 0:p=peek(p1+a):bank 1
   |     :poke(p2+a),p:next
BK | 670 bank 15:p1=p1+5:return
CJ | 680 rem == "save vars" ==
PD | 690 for a=0 to 4:bank 1:p=peek(p1+a):bank 0
   |     :poke(p2+a),p:next
GM | 700 bank 15:p2=p2+5:return
BL | 710 rem == "update zoom parameters" ==
EE | 720 mx=(sx-sq)/2:my=sy-sr:ac=re+(mx*gp)
   |     :bc=im-(my*gp):cc=si/(wi/sd)
CF | 730 gosub 70:gosub 110:return
ME | 740 rem == "disk checks" ==
OE | 750 print#15,"s0:"+fi$:input#15,a,f$:return
CJ | 760 dopèn#1,(fi$)+",p":input#15,a,f$:dclose#1
   |     :return
NP | 770 input#15,a,f$:return
MI | 780 rem ====
EK | 790 rem == "filename" ==
DM | 800 rv$=r1$:gosub 480:kb$=cr$+r$
PI | 810 k=3:gosub 600:f$="":if k=1 then gosub 480
   |     :goto 860
BM | 820 gosub 480:gosub 180:input f$
CM | 830 if f$<>"" then fi$=f$:else f$=fi$
EI | 840 gosub 180:goto800
NC | 850 rem == "pixels" ==
BH | 860 rv$=r1$:gosub 490:kb$=cl$+cr$+"+-"
JH | 870 k=5:gosub 600:on k goto 880,880,890,900
KE | 880 gosub 490:if k=1 then 800:else goto 930
ND | 890 if px=9 then 870:else px=px+1:goto 910
BA | 900 if px=0 then 870:else px=px-1
DL | 910 wd=pa(px,0):gosub 160:goto 870
GI | 920 rem == "plot" ==
LB | 930 rv$=r1$:gosub 500:kb$="  "+cl$+cr$+r$
CM | 940 k=5:gosub 600:on k goto 980,950,950,960
OI | 950 gosub 500:if k=2 then 860:else goto 1000
NF | 960 gosub 500:gosub 290:gosub 360:gosub 260
   |     :gp=si/wd:sys 4864,wd,xc,cn,mo
OI | 970 slow:wi=wd:graphic 4,0,21:gosub 210:zm=0
   |     :gosub 110:goto 930
HB | 980 if wi=0 then 940:else gosub 500:gosub 2010
   |     :goto 930
FI | 990 rem == "load" ==
DO | 1000 rv$=r1$:gosub 510:kb$=cl$+cr$+r$
MI | 1010 k=4:gosub 600:if k=3 then 1030
GO | 1020 gosub 510:if k=1 then 930:else goto 1130
BF | 1030 if fi$="" then f$=fi$:gosub 510:goto 820
GP | 1040 gosub 760:if a<>0 then gosub 510:gosub 180
   |     :goto 790
GA | 1050 fast:bload(fi$),b0:p1=8157:p2=pointer(c1)
   |     :gosub 660
LK | 1060 p2=pointer(c2):gosub 660:p2=pointer(c3)
   |     :gosub 660
LP | 1070 p2=pointer(ac):gosub 660:p2=pointer(bc)
   |     :gosub 660
EG | 1080 p2=pointer(cc):gosub 660:p2=pointer(wi)
   |     :gosub 660
BB | 1090 wd=wi:gp=cc/wd:for a=0 to 9:if wi=pa(a,0)
   |     then px=a
DF | 1100 next:zm=0:gosub 340:gosub 360:gosub 300
   |     :gosub 320
DN | 1110 gosub 270:gosub 210:slow:gosub 510
   |     :goto 930
AE | 1120 rem == "save" ==
JG | 1130 rv$=r1$:gosub 520:kb$=cl$+cr$+r$
HB | 1140 k=4:gosub 600:if k=3 then 1170
```

```
NF 1150 gosub 520:if k=1 then 1000
CI 1160 if mo=0 then 1480:else goto 1280
AO 1170 if fi$="  " then f$=fi$:gosub 520:goto 820
EL 1180 gosub 760:if a=62 then 1210
BH 1190 if a=0 then f$="file exists"
DO 1200 gosub 520:gosub 180:goto 790
DN 1210 fast:gosub 340:p2=8157:for p=0 to 34
       :ma(p)=peek(p2+p):next
KE 1220 p1=pointer(c1):gosub 690:p1=pointer(c2)
       :gosub 690
PM 1230 p1=pointer(c3):gosub 690:p1=pointer(re)
       :gosub 690
OB 1240 p1=pointer(im):gosub 690:p1=pointer(si)
       :gosub 690
LM 1250 p1=pointer(wi):gosub 690:bsave(fi$),b0,p8157
       to p14592:p2=8157
KF 1260 for p=0 to 34:poke p2+p,ma(p):next:slow
       :gosub 520:goto 930
LN 1270 rem == "dump" ==
DA 1280 rv$=r1$:gosub 530:kb$=cl$+cr$+r$
DK 1290 k=4:gosub 600:if k=3 then 1310
IM 1300 gosub 530:if k=1 then 1130:else goto 1480
PG 1310 if fi$="  " then f$=fi$:gosub 530:goto 820
EC 1320 gosub 760:if a=62 then a=px+1:fast
       :goto 1350
NP 1330 if a=0 then f$="file exists"
DH 1340 gosub 530:gosub 180:goto 790
IK 1350 on a gosub 1370,1380,1390,1400,1410,1420,
       1430,1440,1450,1460
LD 1360 slow:goto 1290
BH 1370 bsave(fi$),b0,p28669 to p28928:return
OI 1380 bsave(fi$),b0,p28669 to p29696:return
II 1390 bsave(fi$),b0,p28669 to p30976:return
OI 1400 bsave(fi$),b0,p28669 to p32768:return
HI 1410 bsave(fi$),b0,p28669 to p35072:return
HL 1420 bsave(fi$),b0,p28669 to p37888:return
GI 1430 bsave(fi$),b0,p28669 to p41216:return
CK 1440 bsave(fi$),b0,p28669 to p45056:return
OK 1450 bsave(fi$),b0,p28669 to p49408:return
PL 1460 bsave(fi$),b0,p28669 to p54272:return
DA 1470 rem == "color 1 ==
MM 1480 rv$=r1$:gosub 540:kb$=cl$+cr$+"+-"
AH 1490 k=5:gosub 600:if k>2 then 1520
GN 1500 gosub 540:if k=2 then 1560
DN 1510 if mo=0 then 1130:else goto 1280
OI 1520 if k=3 then c1=c1+1:else c1=c4
FH 1530 if c1=17 then c1=2
AL 1540 gosub 300:goto 1490
GF 1550 rem == "color 2 ==
AC 1560 rv$=r1$:gosub 550:kb$=cl$+cr$+"+-"
OM 1570 k=5:gosub 600:if k>2 then 1590
IP 1580 gosub 550:if k=1 then 1480:else goto 1630
AO 1590 if k=3 then c2=c2+1:else c2=c5
BM 1600 if c2=17 then c2=2
DP 1610 gosub 300:goto 1570
PJ 1620 rem == "color 3 ==
KG 1630 rv$=r1$:gosub 560:kb$=cl$+cr$+"+-"
PA 1640 k=5:gosub 600:if k>2 then 1660
AE 1650 gosub 560:if k=1 then 1560:else goto 1700
CD 1660 if k=3 then c3=c3+1:else c3=c6
NA 1670 if c3=17 then c3=2
MD 1680 gosub 320:goto 1640
OH 1690 rem == "exit" ==
OC 1700 rv$=r1$:gosub 570:kb$=cl$+r$
OC 1710 k=3:gosub 600:if k=1 then gosub 570
       :goto 1630
AF 1720 print chr$(19)chr$(19):graphic 0,1
       :sprite sa(sp,0),0:sys 51938
BJ 1730 if mo=5 then print"switch to 80 col screen"
       :graphic5
IO 1740 return
GF 1750 rem ====
BE 1760 rem == "initialize ==
IK 1770 mo=rgr(0):if mo=5 then print"switch to 40
       col screen"
AH 1780 dim sa(9,1),pa(9,1),ma(34)
FI 1790 zm=0:wi=0:xs=24:ys=50:ac=re:bc=im
       :cc=si:cn=cn and 255
EO 1800 rem = sprite array = sprite# = width =
       index sp
BI 1810 for i=1 to 8:sprite i,0:next
FL 1820 sa(0,0)=1:sa(0,1)=1:sa(1,0)=2:sa(1,1)=2
       :sa(2,0)=3:sa(2,1)=4
FE 1830 sa(3,0)=4:sa(3,1)=6:sa(4,0)=5:sa(4,1)=8
       :sa(5,0)=6:sa(5,1)=10
AI 1840 sa(6,0)=7:sa(6,1)=12:sa(7,0)=5:sa(7,1)=16
       :sa(8,0)=6:sa(8,1)=20
OG 1850 sa(9,0)=7:sa(9,1)=24:sp=4:sn=5:sd=8
JD 1860 rem = pixel array = width = start = index px
ME 1870 px=9:a=16:for i=0 to 9:pa(i,0)=a:a=a+16
       :next
LL 1880 wd=160:a=72:for i=0 to 9:pa(i,1)=a:a=a-8
       :next:px=9
AD 1890 fi$="  ":r$=chr$(13)
MO 1900 rem == "window sprites" ==
FB 1910 color0,1:color4,1:color1,2:graphic1,1
LC 1920 a=1:draw 1,0,0 to 2,0:gosub390
FD 1930 box 1,0,0,4,2:gosub380:box 1,0,0,8,4
       :gosub390
CJ 1940 box 1,0,0,12,6:gosub380:box 1,0,0,16,8
       :gosub390
DE 1950 box 1,0,0,20,10:gosub380:box 1,0,0,23,12
       :gosub390
EO 1960 rem == "graphics screen" ==
CP 1970 c1=3:c2=15:c3=8:gosub 290:graphic 4,1,21
       :slow
OH 1980 gosub 340:gosub 210:gosub 410:gosub 360
       :return
GE 1990 rem ====
OF 2000 rem === "zoom" ===
KG 2010 kb$="  "+cl$+cr$+cu$+cd$+"+-"
       :if zm=0 then mx=wi/2:my=mx:xd=xc:sp=4
EC 2020 zm=1:sr=ys+xd:sq=xs+(xd*2):sx=(mx*2)
       +sq:sy=my+sr
BE 2030 sprite sn,0:sn=sa(sp,0):sd=sa(sp,1)
       :su=xc+wi-sd:ss=su*2
MH 2040 su=ys+su:ss=xs+ss:movspr sn,sx,sy
       :if wi=16 then u=6:else u=9
```

```
ID  2050 movspr sn,sx,sy
EK  2060 k=8:gosub 2150:on k goto 2070,2080,2090,
         2100,2110,2120,2130
OD  2070 c=1:gosub 2230:gosub 720:return
GO  2080 if sx=sq then 2060:else sx=sx-2   :goto 2050
CP  2090 if sx=ss then 2060:else sx=sx+2   :goto 2050
HA  2100 if sy=sr  then 2060:else sy=sy-1  :goto 2050
GB  2110 if sy=su then 2060:else sy=sy+1   :goto 2050
LN  2120 if sp=u  then 2030:else sp=sp+1   :goto 2030
OJ  2130 if sp=0  then 2030:else sp=sp-1   :goto 2030
EE  2140 rem = =  "move window = =
OF  2150 b=0:c=2
JL  2160 get k$:a=1
AJ  2170 do until a=k
GI  2180 if k$=mid$(kb$,a,1) then exit:else a=a+1
GG  2190 loop
AB  2200 gosub 2230:if a<>k then k=a:return
HH  2210 gosub 2250:goto 2160
FH  2220 rem = =
GB  2230 if sp<7 then sprite sn,1,c,0,0,0,0:else sprite
         sn,1,c,0,1,1,0
MN  2240 return
IM  2250 if c=2 then c=1:else c=2
FN  2260 if b=0 then gosub 720:b=1
KP  2270 return
```

Mandelbrot Halo: Creates ML PRG File on Disk

```
BK  100 rem** this program will create the file
        "halo.obj" on disk **
BO  110 rem** for the mandelbrot-set explorer program,
        "halo.bas"  **
PH  120 for i=1 to 540: read x: cs=cs+x: next i
IJ  130 if cs<>56316 then print "checksum error": stop
HO  140 open 1,8,1,"0:halo.obj"
BI  150 print#1,chr$(0)chr$(19);
IL  160 restore
JJ  170 for i=1 to 540: read x: print#1,chr$(x);: next i
PO  180 close 1: end
CD  190 :
GD  200 data 120, 142,  51,  17, 132, 169, 173,  12
EG  210 data  21, 208,  27, 169,  73, 133, 253, 169
DP  220 data  21, 133, 254, 141,  12,  21, 160,  65
DC  230 data 169,   3, 145, 253, 136,  48,   7,  56
HM  240 data 233,   1, 208, 246, 240, 242, 160,   4
EM  250 data 169,   2,  32, 180, 138,  32,  40, 140
MO  260 data 160,  21, 162,  28,  32,   0, 140, 160
DL  270 data   4, 169,   9,  32, 180, 138,  32,  40
OO  280 data 140, 160,  21, 162,  33,  32,   0, 140
MB  290 data 160,   4, 169,  23,  32, 180, 138,  32
LC  300 data  40, 140, 160,  21, 162,  38,  32,   0
CB  310 data 140, 169,   0, 133, 251, 141,  50,  17
IE  320 data 141,  52,  17, 133, 170, 133, 253, 169
BG  330 data 112, 133, 254, 169,  64, 133, 250, 164
KE  340 data 251, 196,   6, 144,  31, 165,   5, 240
IL  350 data  25, 169,  63, 141,   0, 255, 165,   6
KI  360 data 141, 255, 111, 165, 170, 141, 254, 111
FJ  370 data 165, 169, 141, 253, 111, 169,   0, 141
MP  380 data   0, 255,  88,  96, 169,   0,  32, 201
CM  390 data 132, 160,  21, 169,  38,  32,   8, 138
LM  400 data 160,  21, 169,  33,  32,  24, 138, 160
DE  410 data  21, 162,  48,  32,   0, 140, 169,   0
NM  420 data 133, 252, 165,   7, 141,  49,  17, 164
GD  430 data 252, 196,   6, 144,  18, 230, 251, 238
GJ  440 data  51,  17,  24, 165, 253, 101,   6, 133
AA  450 data 253, 144, 164, 230, 254, 208, 160, 169
OI  460 data   0,  32, 201, 132, 160,  21, 169,  38
BB  470 data  32,   8, 138, 160,  21, 169,  28,  32
AE  480 data 137, 138,  32,  72, 136, 160,  21, 162
LC  490 data  43,  32,   0, 140, 160,  21, 169,  13
EB  500 data  32, 212, 139, 160,  21, 162,  53,  32
FE  510 data   0, 140, 160,  21, 162,  58,  32,   0
EO  520 data 140, 160,  21, 162,  63,  32,   0, 140
CA  530 data 160,  21, 162,  68,  32,   0, 140, 160
CJ  540 data   0, 132, 167, 165,   1,  41,  64,  69
FD  550 data 250, 240,  24,  69, 250, 133, 250, 173
PE  560 data  17, 208,  41, 111, 162,   1, 164, 250
DH  570 data 240,   3,   9,  16, 202, 142,  48, 208
PL  580 data 141,  17, 208, 230, 167, 165, 167, 197
BN  590 data   8, 144,   7, 169,   0, 133, 167,  76
OF  600 data 240,  20, 160,  21, 169,  53,  32, 212
JK  610 data 139, 160,  21, 169,  58,  32,   8, 138
LK  620 data 160,  21, 169,  18,  32,   8, 138, 160
II  630 data  21, 169,  48,  32, 137, 138,  32,  72
LG  640 data 136, 160,  21, 162,  58,  32,   0, 140
EC  650 data 160,  21, 169,  68,  32, 212, 139, 160
EJ  660 data  21, 169,  63,  32,  24, 138, 160,  21
BN  670 data 169,  43,  32, 137, 138,  32,  72, 136
HI  680 data 160,  21, 162,  53,  32,   0, 140, 160
FM  690 data  21, 169,  53,  32, 212, 139, 160,  21
FM  700 data 169,  53,  32,   8, 138, 160,  21, 162
EC  710 data  63,  32,   0, 140, 160,  21, 169,  58
PA  720 data  32, 212, 139, 160,  21, 169,  58,  32
DF  730 data   8, 138, 160,  21, 162,  68,  32,   0
CI  740 data 140, 160,  21, 169,  63,  32, 137, 138
HL  750 data  32,  72, 136, 160,  21, 169,  23,  32
FJ  760 data  24, 138,  32,  87, 140,  48,   3,  76
EO  770 data  19,  20, 165, 167, 240,  34, 166,   5
AP  780 data 240,  12, 197, 169, 176,   2, 133, 169
GB  790 data 197, 170, 144,   2, 133, 170, 201,   4
FH  800 data 144,   2,  74,  74, 170, 189,  73,  21
GE  810 data 133, 131,  32,  36, 157,  32,  33, 156
LE  820 data 165,   5, 240,  16, 169,  63, 141,   0
BK  830 data 255, 165, 167, 164, 252, 145, 253, 169
GG  840 data   0, 141,   0, 255, 238,  49,  17, 230
EJ  850 data 252,  76, 183,  19,   0,   0,   0,   0
BD  860 data   0,   0, 130,   0,   0,   0,   0, 131
EK  870 data   0,   0,   0,   0
```

The Last Word On Re–Programming Function Keys

Miklos Garamszeghy
Toronto, ON

Several examples of how to re–program or de–activate the shift–<Run/Stop> and <Help> keys have recently appeared in Transactor's Bits and Pieces section as well as in several other magazine hint columns. Unfortunately, although most of the methods described work adequately under various conditions, they have all missed the very versatile routine built into the C–128 expressly for this purpose: the KERNAL PFKEY routine at 65381 ($FF65). This function is extremely easy to use and is very versatile. Furthermore, its use for redefining any programmable key (including the shift–<Run/Stop> and <Help> keys) does not depend on knowing the length, absolute values or locations for any of the keys. It can be used in the following manner in either program or immediate mode:

1) fill three consecutive zero page locations (such as 250, 251 and 252) with the low byte, high byte and bank of the address of your new key definition text string.

2) set the "a" register to the address of the first zero page location, the "x" register to the number of the key to be defined (from 1 to 10), and the "y" register to the length of the new text string

3) call the routine with a SYS 65381,a,x,y

The length of the new function key definition does not have to be the same as the old one because all function key pointers are automatically updated when you define a new one.

One example is as follows:

```
T$ = "new key definition"
BANK 1: AD = POINTER(T$)
LE = PEEK(AD): LO = PEEK(AD + 1): HI = PEEK(AD + 2)
POKE 250,LO: POKE 251,HI: POKE 252,1
BANK 15: SYS 65381,250,K,LE

          where K =  1 to 8 for F1 to F8
                     9 for shift–<Run/Stop>
                    10 for <Help>
```

To de–activate a key, that is set it to a null string, a simple:

```
BANK 15: SYS 65381,0,K,0
```

is all that is required!!

To restore the original key definitions at the end of your program, the following procedure can be used:

```
POKE 2564,129
```

Followed by:

<Run/Stop>–<Restore>

in immediate mode, or

```
SYS 49275
```

(in either immediate mode or program mode).

Both of these methods activate the KERNAL initialization routine which will also clear screen windows and tab definitions, and reset the colours and set the active screen (i.e. 40 or 80 column) based on the position of the 40/80 key. If bit 3 of memory location 2564 is clear before engaging the routine, the function key definitions will be initialized as part of the process.

While on the subject of function keys, here is a nice little do nothing piece of trivia: how to make function keys activate each other.

Try entering this in immediate mode:

```
KEY 1, "POKE210,0:POKE209,21" + CHR$(13)
```

Then press F1. It will continue to activate itself in an endless loop until you hit <Run/Stop>–<Restore>. The reason is quite simple. BASIC's input editor checks location 209 during keyboard reads. If the value is non–zero, the corresponding number of characters are transferred from the function key buffer area, beginning at the offset location specified in 210, to the input buffer ready for execution. POKEing a non–zero value into 209 will cause the input editor to think that a function key has been pressed. The rest is simple mathematics to figure out the offset into the function key buffer and the number of characters to transfer.

News BRK

Submitting NEWS BRK Press Releases

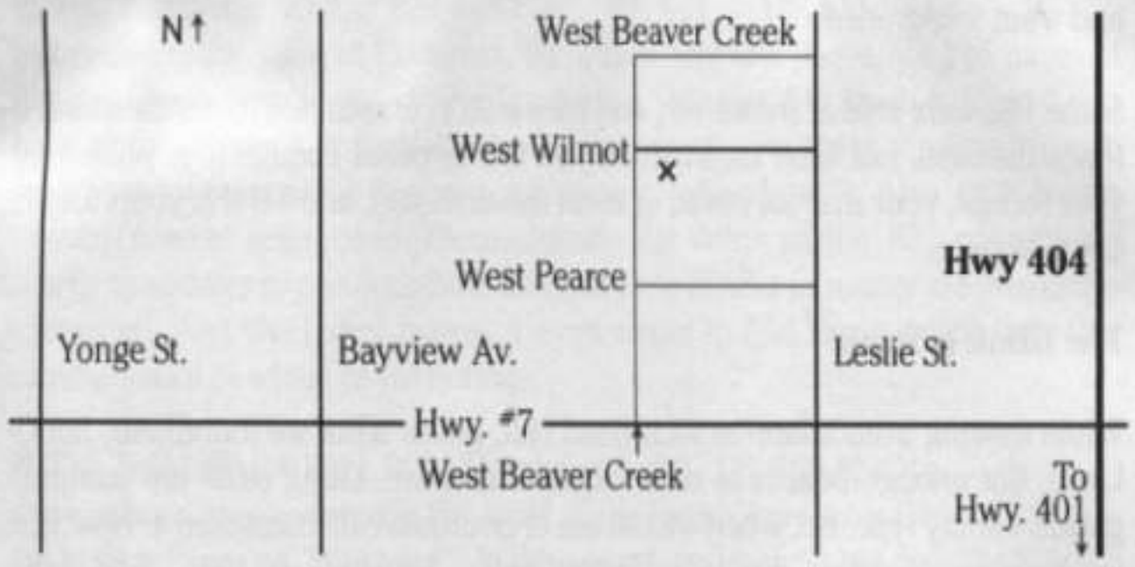
If you have a press release you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Our New Home

Once again we have a new address. Actually, our last new address was just our post office box down the street. This new address is our new headquarters in the Beaver Creek business park of Richmond Hill.

The Transactor
85 West Wilmot Street, Unit #10
Richmond Hill, Ontario Canada
L4B 1K7
phone (416) 764-5273



If you're in the neighborhood, drop in on us for the grand tour! We're easy to find. From the Toronto area, take Hwy 401 to Hwy. 404, go north to Hwy. 7, west to Leslie Street, go north two streets and that's West Wilmot (notice, no 'n' in 'Wilmot'). Go west and we're the last building on the left (red bricks, dark green garage doors). Just past us there's a big empty lot so we may not be the last building on the left for long.

Advertisers Wanted

If anyone is interested in placing full-page, half-page or quarter-page colour or black and white ads in the Transactor, please contact us for rates and information. Yup, you heard right. We'll take ads now, but space is limited. Our ceiling currently is the cover spots plus 5 pages of the interior.

New Canadian Prices

In an act of boundless generosity and financial miscalculation, we have until now given our Canadian customers a real break by pricing products the same in Canadian and U.S. dollars. We felt good about giving our fellow Canadians a break, but that feeling is quickly giving way to the bad feelings we're getting about losing money. So, we have adjusted (raised) prices for products when paid in Canadian currency. It's also the first time in 7 years that subscription prices have gone up.

With the exception of the Micro Sleuth diagnostic device, the U.S. prices remain the same. You will see the new prices on the subscription card, but here they are so you don't get taken by surprise:

Magazine Subscriptions	19.00 Canadian
Magazine Cover Price	4.25 Canadian
Disk Subscriptions	55.00 Canadian
Transactor Disks	9.95 Canadian
The Bits and Pieces Book	17.95 Canadian
Inner Space Anthology	17.95 Canadian
1541 ROM Upgrade Kit	69.95 Canadian
T-Shirts S – XL	15.95 Canadian
Jumbo T-Shirt	19.95 Canadian

As mentioned above, the Micro Sleuth is an exception; the U.S. price goes from $89.95 to $99.95 U.S., while the price in Canadian dollars remains the same.

Many of the price increases do not reflect the exchange rate exactly, but shipping to the U.S. is more expensive, and there's also brokerage fees.

Cover Price Increase

Our cover price in Canada is up to $4.25, another good reason to subscribe. Although we're no longer on the newsstand, some are still being shipped to the odd computer store, but in very few places. But even if you can buy them locally off a magazine rack, it's 34% more expensive than subscribing, 40% more in the US.

Shipping Fee on Mail Orders

We have added a small shipping/handling flat rate to all mail orders. With ANY order, please add $2.00 Cdn. within Canada, $2.00 US in the states, and $5.00 US for foreign orders. This does NOT apply to back issues (shipping costs are already built into the price of back issues) or subscription orders (i.e. magazine and/or disk subscriptions).

Don't Forget the Sales Tax!

If you are a resident of Ontario, please don't forget to add the 7 percent sales tax to all orders, including disk subscriptions. There is no tax on magazine subscriptions or books, but the tax applies to EVERYTHING else.

Sign Of The Times

We get many orders in on our postage paid order card that show a Visa or Master Card number. Each time we must call Visa or M/C to get a verification of the card number, expiry date, etc, even for small amounts. Why are gas stations and department stores not required to do this? One reason: a signature. That signature means the person making the purchase is the same person who owns the card, at least in principle. If the card is not on the "hot list" and we have a signature, many orders won't need verification, which will save us hours, maybe days! And those days add up to late shipments which you enjoy about as much as we do.

You can help. Our order card now has a space for a signature. When using your credit card for payment, please sign, and be sure to indicate the date it was written. That way your signature is only good for that particular card. Visa and M/C want copies of our card submitted to match up with our invoicing, and if the difference between the date on your order card and our invoice date is too big, we'll be called to the question stand. . . and we don't want that.

Dealer Inquiries Welcome

The Transactor has several products besides our magazine and disk: the Bits and Pieces book, the Inner Space Anthology, the TransBASIC disk, the Potpourri disk (see ad this issue), and the Micro Sleuth. These products are currently being

marketed and sold through the magazine only. We would be happy to bring these products to a larger audience by selling to any interested dealers; if you are one of them, please contact us.

Group Subscription Rates: The 20/20 Deal

The Transactor has always been popular among Commodore user groups, so to encourage new subscribers we are offering quantity discounts for magazine and disk subscriptions: 20 percent off for group orders of 20 or more subscriptions. If you can get together enough friends or club members, just put all the subscriptions in a single envelope, and you get the discount. You don't need to be a user group to qualify – any 20 or more subscription cards in a single package get the 20/20 deal, no questions asked.

T–Shirt Offer Continues

Y'know, I just can't believe these T-Shirts. They were ordered from Vantage Sports here in Toronto, and if anyone else around southern Ontario is planning to get some made, Vantage is *the* place to go. Their prices are a little higher, but the shirts are well worth it. Make sure to specify the "super opaquing process" if you're getting something screened onto them and they may just last forever. We've had the T's now for almost a year. I started out with two last July, and recently had to "borrow" one from stock to wear to a special event when going home to change would have taken me somewhat out of my way. Now I can't tell the difference between my newest one and the other two which are easily 10 months older.

Order a combination magazine AND disk subscription, and one of these fabulous T-Shirts will be sent to you FREE. Please indicate the size you want (sorry, Jumbo excluded) and the color on the order card. Before now the shirts came in red only. Now we have **red and blue!** The front features our mascot, Duke, in a snappy white tux and top hat, standing behind our logo in 3D letters.

Mail–Order Products No Longer Offered

We have removed several products from our mail–order card: The Gnome Speed Compiler and Gnome Kit Utility, the "pocket" series of software, PRISM's SuperKit 1541, the BH100 hardware course material, the Anchor Volksmodems, and the Comspec 2 megabyte RAM expansion units. We still have some stock of the software and can order more of the other products if necessary, so we should be able to fill any orders from previous subscription cards.

New Mail–Order Products

Now the good news. As you can see on the mail–order card, we have four new Transactor products to offer you:

The Bits and Pieces Disk: This disk contains all of the programs from the Transactor book of Bits and Pieces (the "bits book"), which in turn come from the "Bits and Pieces" section of past issues of the magazine. The "bits disk" can save you a lot of typing, and in conjunction with the bits book and its comprehensive index can yield a quick solution to many a programming problem. Price for the disk is the same as our regular disks, $8.95 US, 9.95 Cdn.

Bits Book AND Disk: Get both for just $19.95 US, 24.95 Cdn.

The Amiga Disk is here! Finally, the first Transactor Amiga disk is available. It contains all of the Amiga programs presented in the magazine, of course, including source code and documentation. You will find the popular "PopColours" program, the programmer's companion "Structure Browser", the Guru–killing "TrapSnapper", user–friendly "PopToFront", and others. In addition, we have included public domain programs – again, with documentation – that we think Transactor readers will find useful. Among these are the indispensable ARC; Csh, a powerful CLI–replacement DOS shell; BLink, a linker that is much faster and has more features than the standard ALink; Foxy and Lynx, a 6502 cross assembler and linker that makes its debut on the Amiga Disk; and an excellent shareware text editor called UEdit. In addition, we have included our own expression–evaluator calculator that uses variables and works in any number base. All programs contain source code and documentation; all can be run from the CLI, and some from Workbench. There's something for everyone on the Transactor Amiga disk. Price is $12.95 US, $14.95 Canadian.

The Potpourri Disk: This is a C-64 product from the software company called AHA!, otherwise known as Nick Sullivan and Chris Zamara. The Potpourri disk is a wide assortment of 18 programs ranging from games to educational programs to utilities. All programs can be accessed from a main menu or loaded separately. No copy protection is used on the disk, so you can copy the programs you want to your other disks for easy access. Built–in help is available from any program at any time with the touch of a key, so you never need to pick up a manual or exit a program to learn how to use it. Many of the programs on the disk are of a high enough quality that they could be released on their own, but you get all 18 on the Potpourri disk for just $17.95 US / $19.95 Canadian. See the Ad in this issue for more information.

TransBASIC II

An updated TransBASIC disk is now available, containing all TB modules ever printed. The first TransBASIC disk was released just as we published TransBASIC Column #9 so the modules from columns 10, 11 and 12 did not exist. The new manual contains everything in the original, plus all the docs for the extras.

Prices for the new TB disk are $17.95 US and $19.95 Cdn. People who ordered TB I can upgrade to TB II for the price of a regular Transactor Disk (8.95/9.95). If you are upgrading, you don't necessarily need to send us your old TB disk; if you ordered it from us, we will have your name on file and will send you TB II for the upgrade price. Please indicate on the order form that you have the original TB and want it upgraded.

Some TBs were sold at shows, etc, and they won't be recorded in our database. If that's the case, just send us anything you feel is proof enough (e.g. photocopy your receipt, your manual cover, or even the diskette), and TB II is yours for the upgrade price.

The Glink is Back!

While moving from Milton to Richmond Hill, guess what we found? No, not G-Links, but enough boards to make about 200 more. Glink parts are common garden variety type, but when we ran out of boards we discontinued it. Now that we have more, we've decided to make more. Too bad we didn't find them sooner. . . many orders for this item had to be denied. However, we were surprised to find that many of the parts needed have had price increases since we discontinued it. Regardless, they're still the least expensive interfaces around. Glinks are $59.95 US, 69.95 Cdn.

The Glink is a Commodore 64 to IEEE interface. It allows the 64 to use IEEE peripherals such as the 4040, 8050, 9090, 9060, 2031, and SFD-1001 disk drives, or any IEEE printer, modem, or even some Hewlett-Packard and Tektronics equipment like oscilloscopes and spectrum analyzers. The beauty of the Glink is its "transparency" to the C64 operating system. Some IEEE interfaces for the 64 add BASIC 4.0 commands and other things to the system that sometimes interfere with utilities you might like to install. The Glink adds nothing! In fact it's so transparent that a switch is used to toggle between serial and IEEE modes, not a linked-in command like some of the others. Switching from one bus to the other is also possible with a small software routine as described in the documentation.

As of Transactor Disk #19, a modified version of Jim Butterfield's "COPY–ALL" will be on every disk. It allows file copying from serial to IEEE drives, or vice versa.

New Set of Microfiche

Some of our back issues are not available any more, but they're all available on microfiche. Since we're now into Volume 8, a set of microfiche will include all issues from Volume 4 through Volume 7. Prices are $49.95 U.S and $59.95 Cdn.

Transactor Mail Order

The following details are for products listed on the mail order card. If you have a particular question about an item that isn't answered here, please write or call. We'll get back to you and most likely incorporate the answer into future editions of these descriptions so that others might benefit from your enquiry.

■ Moving Pictures - the C-64 Animation System, $29.95 (US/C)
This package is a fast, smooth, full-screen animator for the Commodore 64, written by AHA! (Acme Heuristic Applications!). With Moving Pictures you use your favourite graphics tool to draw the frames of your movie, then show it at full animation speed with a single command. Movie 'scripts' written in BASIC can use the Moving Pictures command set to provide complete control of animated creations. BASIC is still available for editing scripts or executing programs even while a movie is being displayed. Animation sequences can easily be added to BASIC programs. Moving Pictures features include: split screen operation – part graphics, part text – even while a movie is running; repeat, stop at any frame, change position and colours, vary display speed, etc; hold several movies in memory and switch instantly from one movie to another; instant, on-line help available at the touch of a key; no copy protection used on disk.

■ Transactor T-Shirts, $13.95 US, $15.95 Cdn.
■ Jumbo T-Shirt, $17.95 US, $19.95 Cdn.
As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. The Jumbo makes a good night-shirt/beach-top – it's BIG. I'm 6 foot tall, and weigh in at a slim 150 pounds – the Small fits me tight, but that's how I like them. If you don't, we suggest you order them 1 size over what you usually buy.

■ The Transactor Book of Bits and Pieces #1, $14.95 US, $17.95 Cdn.
Not counting the Table of Contents, the Index, and title pages, it's 246 pages of Bits and Pieces from issues of The Transactor, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference – no more flipping through magazines for that one bit that you just know is somewhere. . . Also, each item is forward/reverse referenced. Occassionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Tr@ns@ctor 1541 ROM Upgrades, $59.95 US, $69.95 Cdn.
You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. There are 2 ROMs per set, and they fix not only the SAVE@ bug, but a number of other bugs too (as described in P.A. Slaymaker's article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn't been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

NOTE: Our ROM upgrade kit does NOT fit in the 1541C drives. Where we supply two ROMs, Commodore now has it down to one MASSIVE 16 Kbyte ROM. We don't know if the new drives still contain the bugs eliminated by our kit, but we'll find out and re-cut a second kit if necessary. In the meantime, 1541C owners should not order this item until further notice.

■ The Micro Sleuth: C64/1541 Test Cartridge, $99.95 (US), $129.95 (Cdn)
This cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:

1) Check drive speed
2) Check drive alignment
3) 1541 Serial test
4) C64 serial test
5) Joystick port 1 test
6) Joystick port 2 test
7) Cassette port test
8) User port test

A second board, that plugs onto the User Port, contains 8 LEDs that lets you zero in on the faulty chip. Complete with manual.

■ Inner Space Anthology $14.95 US, $17.95 Cdn.
This is our ever popular Complete Commodore Inner Space Anthology. Even after a year and a half, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no "reading" material (ie. "paragraphs"). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ The TransBASIC Disk II $17.95 US, $19.95 Cdn.
This is the complete collection of every TransBASIC module ever published. There are over 140 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

Transactor Disks, Transactor Back Issues, and Microfiche

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issues are ONLY available on microfiche – these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, both for single copies and subscriptions, with one exception: a complete set of 24 (Volumes 4, 5, 6, and 7) will cost just $49.95 US, $59.95 Cdn.

This list also shows the "themes" of each issue. "Theme issues" didn't start until Volume 5, Issue 01. The Transactor Disk #1 contains all programs from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1–3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled-down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

■ Vol. 4, Issue 01 (■ Disk 1)
■ Vol. 4, Issue 02 (■ Disk 1)
■ Vol. 4, Issue 03 (■ Disk 1)
■ Vol. 4, Issue 04 – MF only (■ Disk 1)
■ Vol. 4, Issue 05 – MF only (■ Disk 1)
■ Vol. 4, Issue 06 – MF only (■ Disk 1)
■ Vol. 5, Issue 01 – Sound and Graphics (■ Disk 2)
■ Vol. 5, Issue 02 – Transition to Machine Language – MF only (■ Disk 2)
■ Vol. 5, Issue 03 – Piracy and Protection – MF only (■ Disk 2)
■ Vol. 5, Issue 04 – Business & Education – MF only (■ Disk 3)
■ Vol. 5, Issue 05 – Hardware & Peripherals (■ Disk 4)
■ Vol. 5, Issue 06 – Aids & Utilities (■ Disk 5)
■ Vol. 6, Issue 01 – More Aids & Utilities (■ Disk 6)
■ Vol. 6, Issue 02 – Networking & Communications (■ Disk 7)
■ Vol. 6, Issue 03 – The Languages (■ Disk 8)
■ Vol. 6, Issue 04 – Implementing The Sciences (■ Disk 9)
■ Vol. 6, Issue 05 – Hardware & Software Interfacing (■ Disk 10)
■ Vol. 6, Issue 06 – Real Life Applications (■ Disk 11)
■ Vol. 7, Issue 01 – ROM / Kernel Routines (■ Disk 12)
■ Vol. 7, Issue 02 – Games From The Inside Out (■ Disk 13)
■ Vol. 7, Issue 03 – Programming The Chips (■ Disk 14)
■ Vol. 7, Issue 04 – Gizmos and Gadgets (■ Disk 15)
■ Vol. 7, Issue 05 – Languages II (■ Disk 16)
■ Vol. 7, Issue 06 – Simulations and Modelling (■ Disk 17)
■ Vol. 8, Issue 01 – Mathematics (■ Disk 18)
■ Vol. 8, Issue 02 – Operating Systems (■ Disk 19)

Industry News

The following items, compiled by Astrid Kumas, are based on press releases recently received from the manufacturers. Please note that product descriptions are not the result of evaluation by The Transactor.

Portland Company Vanishes

News BRK in Volume 7, Issue 6 carried an item about a video digitizer named Eye-Scan for the Commodore 64 from a company named Digital Engineering and Design in Portland, Oregon. It seems that Digital Engineering has either moved or folded, as neither we nor several readers who have tried have been able to get in touch with them. If you *are* out there somewhere, Digital, let us know where you went. We might have some customers for you.

4040 Drive Internals

Depending on reader response, a book could soon become available that uncovers, for the very first time, all inner details of the Commodore 4040 drive. Within this vast tome of knowledge will be found an in depth and documented look into the Floppy Disk Controller RAM and ROM, the Interface Processor RAM and ROM, plus theory on how it all fits together. A useful book for specific occasions. The book is close to completion right now, but reader response is required to determine if full production would be worth while. If you are at all interested, and would like to be kept informed of the book's progress, then send a note today to the following address. If the 4040 book is successful, then an 8050, 8250, 9060 and 9090 will follow.

Hilaire Gagne
1074 Webbwood Drive
Sudbury, Ontario, Canada
P3C–3B7

CAD for the Amiga

On November 17, 1986 Aegis Development began to ship their latest Amiga product, Aegis Draw Plus, to dealers and distributors. This new computer–aided design package for the Amiga allows up to six independent drawings of 256 layers each to be worked on using a basic 512K Amiga computer (although one megabyte of RAM is recommended). Full 16–colour capability is available and drawings may be saved in the Amiga's standard IFF file format for use in other programs such as Aegis Images, Graphicraft, and Deluxe Paint paint programs.

Aegis Draw Plus is controlled either with the mouse and pull–down menus, or entirely with the keyboard for more advanced users. Some of the capabilities beyond the ability to draw lines and shapes include:

Basic features:

- ruler lines with variable measure types (decimal, feet, inches, etc.)
- adjustable grid sizes and on/off toggle
- plotter drivers selectable via menu for use with multiple plotters
- advanced printer support for clean dot matrix output (72 dpi)
- unlimited levels of zoom
- variable line weights and fill patterns, including solid fills
- full 360 degree rotation of any object or part
- resizing of any object or part
- 256 selectable levels (planes) to work on
- adjustable and savable color palette
- text can be typed directly on any part of any display
- multiple resolution (640x400 – 1mb RAM required – and 640x200)
- file compatibility between resolutions
- eight–way mirror function.

Advanced features:

- parts library for storage of often–used objects
- "stats" function allows precise numeric adjustment of any item
- hook tool for distorting polygons (as found in Aegis Animator)
- array tool for creating repeated objects in a pattern
- function key support for toolbox selection
- arcs allow variance of angle and radius in a single operation
- automatic dimensioning and scaling
- plot spooling
- plot files can be saved to disk for reprinting
- fully 1.2 DOS compatible
- grid size and rounding consistent in zoom operation
- locked font sizes (adjustable via stats)
- customizable plotter driver for any hardware–compatible plotter
- not copy protected for easy transfer to hard drive
- files are compatible between Aegis Draw and Aegis Draw Plus
- works with Genlock, digitizers, track balls, expanded memory (up to 8 megabytes), and hard disks.

Aegis Draw Plus retails for $259.95 (US). Registered users who own Aegis Draw, the company's first design program for the Amiga, will receive notification of Aegis' update policy.

For further information on Aegis Draw Plus, contact:

Aegis Development, Inc.
2210 Wilshire Blvd. #277
Santa Monica CA 90403
(213) 306–0735

B.E.S.T.Business Management

B.E.S.T. Inc. (Business Electronics Software & Technology, Inc.) announces B.E.S.T. Business Management for the Amiga computer, an accounting/business information management software system that includes Order Processing, Inventory Management, Services Management, Accounts Receivable, Accounts Payable and General Ledger. Special features of the program listed by the manufacturer allow the user to:

- create and save as many as fifteen customized financial reports;
- select from fourteen preformatted Inventory reports, or create and save up to fifteen unique Inventory reports, from a menu of 33 inventory performance factors;
- define and manage, by invoice or by customer, nine different sets of "Terms and Conditions" of sale;
- define and categorize Services, units of service and fees per service unit, and "bill" labour charges or "no charge" warranty services;
- automatically update Inventory, Receivables and Ledger accounts when a business procedure is completed;
- manage multiple sales/excise tax requirements.

The package includes a 380–page owners manual, containing 260 screen photographs and 50 sample reports. For further information regarding price and availability, contact:

B.E.S.T., Inc.
P.O. Box 230519
Tigard, OR 97224
(503) 684–6655
1–800–368–BEST

Public Domain Programs

Two US–based sources of public domain software have recently come to our attention. They are the Schneider Software Company and the Folklife Terminal Club.

Schneider Software at 23 East Green St., West Hazleton, PA 18201 sells Frugalware, public domain software for the Commodore 64/128 and the Commodore Amiga. Three hundred disks containing over 8000 programs presently run on the C–64 and C–128. Some categories include Games, Utilities, Business, Graphics and Music.

The price per disk is $2.50 (US), not including quantity discounts. The Public Domain Catalogue (on disk) and a free disk containing a word processor, a

database and a spreadsheet plus thirty additional programs can be obtained for $2.00 (US) postage and handling.

From the same company, twenty-five disks are available for the Amiga. The price is $4.95 (US) per disk plus $2.00 (US) for postage and handling.

Folklife Terminal Club, an international Commodore computer users group, provides support for the Plus/4, VIC 20, PET, CBM, B-128, C-64 and C-128 computers. The club has issued new catalogues of software from their archives, which contain more than 6000 user written programs in the areas of Education, Science, Business, Games, Music, Graphics and more than twenty-five other categories.

The programs are stored on diskettes and are usable on various configurations of the orphaned computers as well as the current C-64 and C-128 machines. The software itself is free. The first diskette that should be ordered is the new Catalogue On A Disk which contains an automatic software finder program, a listing of all the available software in the Folklife library, complete instructions and Associate Membership in the club. There is a copying and mailing fee of $15.00 (US) per diskette. Use bank-issued cheques payable on a US bank or Post office International Money Orders. There is a separate Catalogue Disk for each of the Commodore computers, so specify which computer and disk drive you have. Contact:

Folklife Terminal Club
Box 555-HN
Co-op City Station
Bronx, NY 10475

The New PAL JR.

Byte By Byte Corporation has announced that their product for the Amiga computer, PAL JR, has been completely redesigned.

The PAL JR is a two-slot, fully Zorro compatible auto-configure expansion system. The standard PAL JR system contains 1 MByte of fast RAM, a battery backed clock calendar, and a 20 MByte hard disk drive with DMA controller. The DMA controller occupies one slot and will support a SCSI option. The PAL JR system is contained in a low profile case designed to sit on top of Amiga.

Shipment of the PAL JR will start in the first quarter of 1987. Pre-paid orders will be given preference when shipping commences. The price will be $1495.00 US, and all orders will be filled directly by Byte By Byte. For additional information, or to place an order, contact:

Byte By Byte Corp.
Arboretum Plaza II
9442 Capital of Texas Highway North
Suite 150, Austin TX 78759
(512)343-4357

NLQ for the Gemini 10X

Chessoft Ltd. has developed software-controlled near letter quality print for the Gemini 10X, Commodore 64, 1541 home computer system.

Their product, Gem-LQ, operates on ordinary sequential files, which can be prepared directly with most word processors. It accepts either true ASCII or Commodore codes. The user can modify any character, or prepare completely new customized character sets.

Gem-LQ is available exclusively from Chessoft Ltd. for $29.95 (US) ppd. plus $3.00 (US) for overseas orders. For an original printout sample and further information, interested C-64 users are invited to send $1.00 (US - refundable) to:

Chessoft Ltd.
723 Barton St.
Mt. Vernon
IL 62864

Supradrive Amiga Hard disk

Supra Corporation has announced the release of SupraDrive hard disk systems for the Amiga computer. The SupraDrive system includes four integrated features: hard disk drive, real-time clock with battery backup for time and date retention, SCSI expansion port, and the capability to expand the Amiga's RAM memory.

SupraDrives are available in 20, 30, and 60 mb capacities and come ready to plug-in and use. The retail prices are $995, $1195, and $1995 (US) respectively.

The SupraDrive plugs onto the Amiga's expansion connector and features Supra's own proprietary interface for high-speed data transfers. The data channel is capable of burst data transfers of over 250 KB per second.

The SupraDrive interface has the capability of adding plug-in RAM modules with capacities from 512K to 4 megabytes of Fast Ram. The expansion RAM boards and hard disk are powered by the SupraDrive's own power supply. For more information call:

John Wiley (503)967-9075
1133 Commercial Way
Albany, OR 97321

Auto Disk Menu/Program Loader

Autoload, a disk file directory and loading utility, is now available from Southern System Services for the Commodore 64, 64C, 128, SX-64 and DX-64 computers with the 1541 or 1571 (or equivalent) disk drive.

When saved as the first program on disk, Autoload provides two keystroke disk directory screen listing for up to 100 disk files, single keystroke file load and run, forward and reverse file listing window scroll, function key exit to BASIC with single keystroke directory program restart and function key directory reload.

Autoload is completely menu driven, and can be customized to list only boot files and to operate or interact with drive addresses 8, 9, 10 and 11. Deleting itself from the directory listing, Autoload is transparent to the user.

The unprotected program costs $18.00 (US). Make cheque or money order payable to:

Southern System Services
1307 Krenek
Crosby, TX 77532
(713) 328-3451

A-Talk Communication Tools for the Amiga

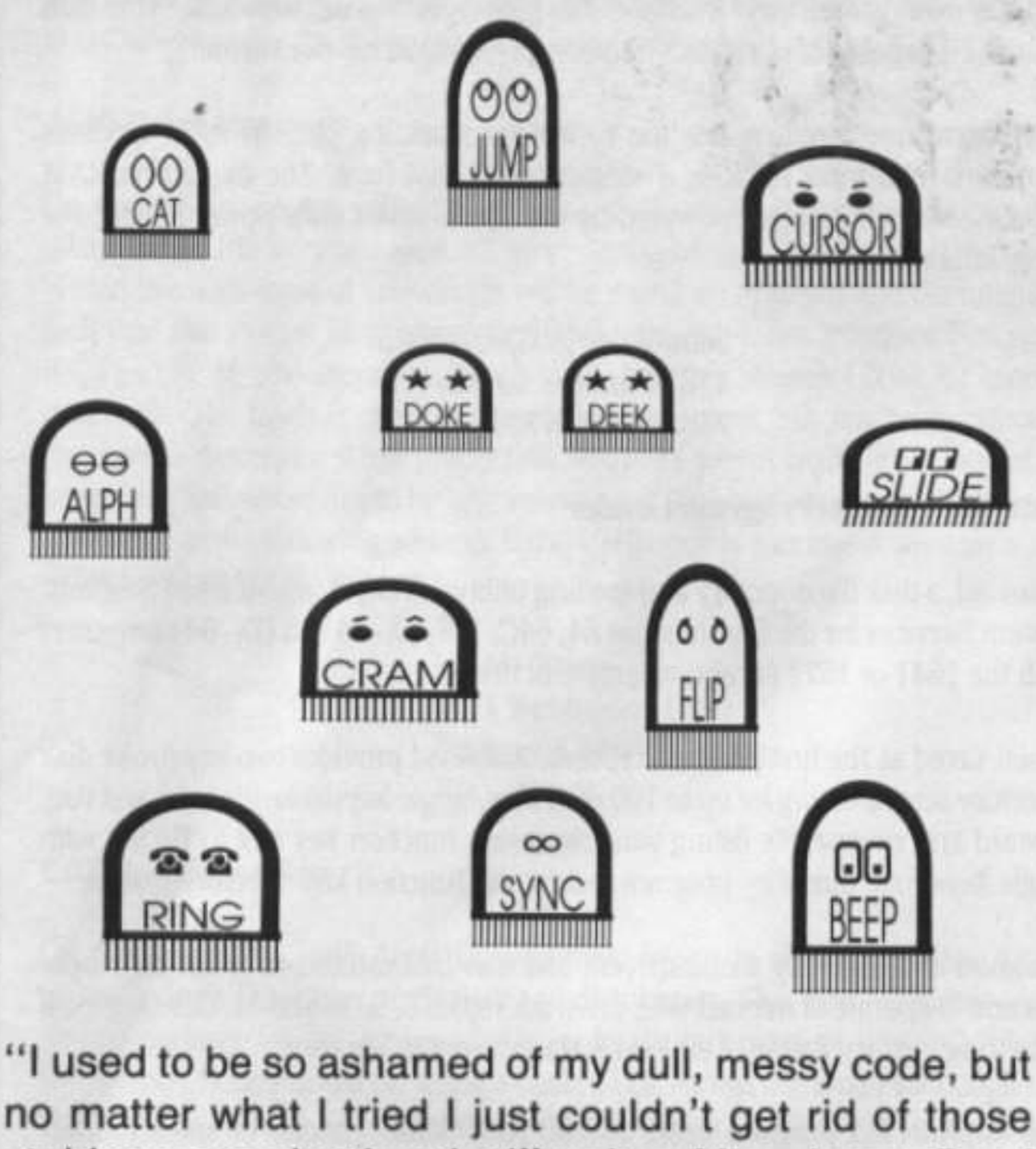
Felsina Software announces the release of A-Talk, an advanced communication and terminal program for the Amiga. A-Talk is a set of communication tools that work together to help you collect, control, and transmit data with your Commodore Amiga. A-Talk supports Kermit, Xmodem and Compuserve "B" error-checking protocols, as well as allowing transfer of standard ASCII files. A script language called "Dial-Talk" allows you to automate your login process and includes a phone directory and programmable function keys. Standard Login scripts are included for connecting with various networks.

Full ANSI terminal emulation is supported, and Termcap and terminfo descriptions are included to allow use of full-screen editors like Emacs and vi on UNIX systems. A-TALK for the Amiga list for $49.95. A-TALK is NOT copy protected.

Felsina Software Inc.
3175 S.Hoover Street, Suite 275
Los Angeles, California 90007
(213) 747-8498

New! Improved! TRANSBASIC 2!

with SYMASS™



"I used to be so ashamed of my dull, messy code, but no matter what I tried I just couldn't get rid of those stubborn spaghetti stains!" writes Mrs. Jenny R. of Richmond Hill, Ontario. "Then the Transactor people asked me to try new TransBASIC 2, with Symass™. They explained how TransBASIC 2, with its scores of tiny 'tokens', would get my code looking clean, fast!

"I was sceptical, but I figured there was no harm in giving it a try. Well, all it took was one load and I was convinced! TransBASIC 2 went to work and got my code looking clean as new in seconds! Now I'm telling all my friends to try TransBASIC 2 in *their* machines!"

• • • • • • •

TransBASIC 2, with Symass, the symbolic assembler. Package contains all 12 sets of TransBASIC modules from the magazine, plus full documentation. Make your BASIC programs run faster and better with over 140 added statement and function keywords.

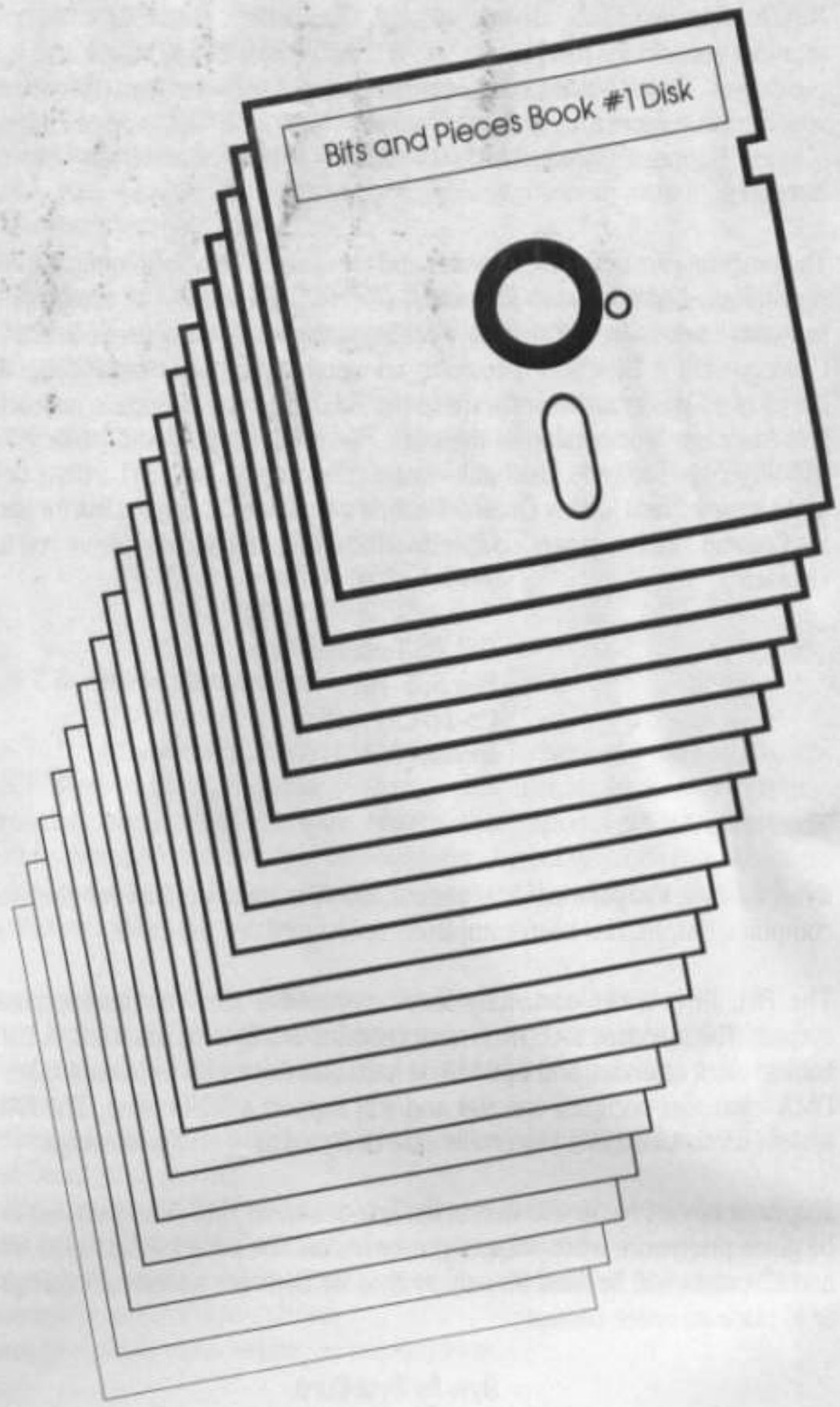
Disk and Manual $17.95 US, $19.95 Cdn.

(see order card at center and News BRK for more info)

TransBASIC 2

"Cleaner code, load after load!"

Bits & Pieces I: The Disk



From the famous book of the same name, Transactor Productions now brings you *Bits & Pieces I: The Disk!* You'll **thrill** to the special effects of the screen dazzlers! You'll **laugh** at the hours of typing time you'll save! You'll be **inspired** as you boldly go where no bits have gone before!

"Extraordinarily faithful to the plot of the book... The BAM alone is worth the price of admission!"
Vincent Canbyte

"Absolutely magnetic!!"
Gene Syscall

"If you mount only one bits disk in 1987, make it this one! The fully cross-referenced index is unforgettable!
Recs Read, New York TIS

WARNING: Some sectors contain null bytes. Rated GCR

BITS & PIECES I: THE DISK, A Mylar Film, in association with Transactor Productions. Playing at a drive near you!

Disk $8.95 US, $9.95 Cdn. Book $14.95 US, $17.95 Cdn.
Book & Disk Combo Just $19.95 US, $24.95 Cdn!

The Potpourri Disk

Help!

This HELPful utility gives you instant menu-driven access to text files at the touch of a key - while any program is running!

Loan Helper

How much is that loan really going to cost you? Which interest rate can you afford? With Loan Helper, the answers are as close as your friendly 64!

Keyboard

Learning how to play the piano? This handy educational program makes it easy and fun to learn the notes on the keyboard.

Filedump

Examine your disk files FAST with this machine language utility. Handles six formats, including hex, decimal, CBM and true ASCII, WordPro and SpeedScript.

Anagrams

Anagrams lets you unscramble words for crossword puzzles and the like. The program uses a recursive ML subroutine for maximum speed and efficiency.

Life

A FAST machine language version of mathematician John Horton Conway's classic simulation. Set up your own 'colonies' and watch them grow!

War Balloons

Shoot down those evil Nazi War Balloons with your handy Acme Cannon! Don't let them get away!

Von Googol

At last! The mad philosopher, Helga von Googol, brings her own brand of wisdom to the small screen! If this is 'AI', then it just ain't natural!

News

Save the money you spend on those supermarket tabloids - this program will generate equally convincing headline copy - for free!

Wrd

The ultimate in easy-to-use data base programs. WRD lets you quickly and simply create, examine and edit just about any data. Comes with sample file.

Quiz

Trivia fanatics and students alike will have fun with this program, which gives you multiple choice tests on material you have entered with the WRD program.

AHA! Lander

AHA!'s great lunar lander program. Use either joystick or keyboard to compete against yourself or up to 8 other players. Watch out for space mines!

Bag the Elves

A cute little arcade-style game; capture the elves in the bag as quickly as you can - but don't get the good elf!

Blackjack

The most flexible blackjack simulation you'll find anywhere. Set up your favourite rule variations for doubling, surrendering and splitting the deck.

File Compare

Which of those two files you just created is the most recent version? With this great utility you'll never be left wondering.

Ghoul Dogs

Arcade maniacs look out! You'll need all your dexterity to handle this wicked joystick-buster! These mad dog-monsters from space are not for novices!

Octagons

Just the thing for you Mensa types. Octagons is a challenging puzzle of the mind. Four levels of play, and a tough 'memory' variation for real experts!

Backstreets

A nifty arcade game, 100% machine language, that helps you learn the typewriter keyboard while you play! Unlike any typing program you've seen!

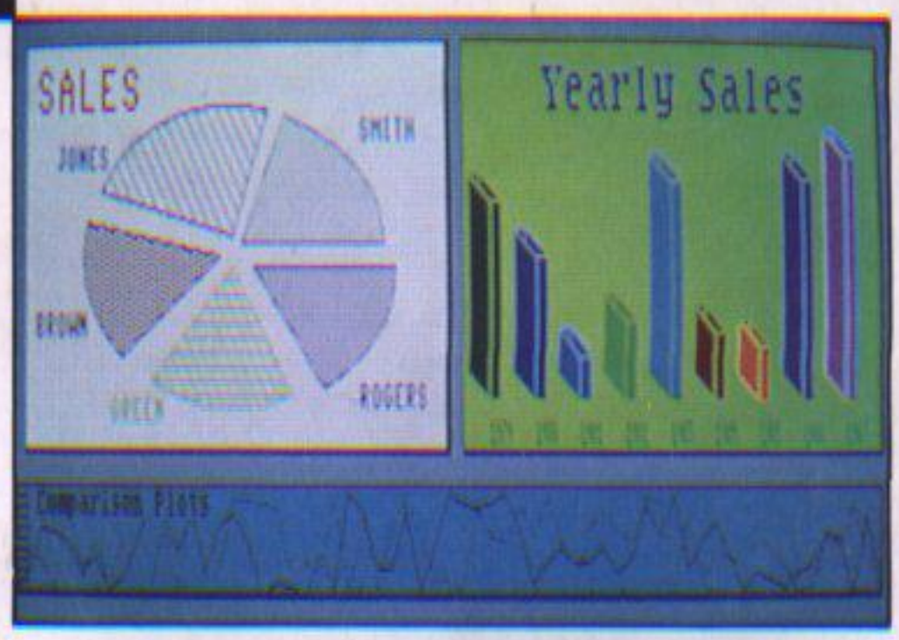
All the above programs, just $17.95 US, $19.95 Canadian. No, not EACH of the above programs, ALL of the above programs, on a single disk, accessed independently or from a menu, with built-in menu-driven help and fast-loader.

The ENTIRE POTPOURRI COLLECTION
JUST $17.95 US!!

See Order Card at Center

Introducing BASIC 8

By Lou Wallace & David Darus



At last, you can unleash the graphics potential of your Commodore 128 to achieve performance which rivals that of 16-bit micros! Imagine your 128 (or 128-D) producing resolution of 640 x 200 in monochrome and 640 x 192 in 16 colors without any additional hardware. Sound impossible? Not with **Basic 8**, the new graphics language extension.

Basic 8 adds over 50 new graphics commands to standard C-128 Basic. Just select one of many graphics modes and draw 3-D lines, boxes, circles and a multitude of solid shapes with a single command. We've even added commands for windows, fonts, patterns and brushes.

To demonstrate the power and versatility of this new graphics language, we have created **Basic Paint**, a flexible icon-based drawing application. Written in **Basic 8**, **Basic Paint** supports an expanded Video RAM (64K), RAM Expanders, Joystick and the New 1351 Proportional Mouse.

Also included is an icon-based desk-top utility which provides quick and convenient access to each of your very own **Basic 8** creations.

All this graphics potential is yours at the special introductory price of $39.95. The package includes **Basic 8**, **Basic Paint**, the desk-top utility, a 180-page manual and a run time module. (80-Column RGB Monitor Required)

WIN $1000.* PROGRAMMING CONTEST

Mail your order to:
Computer Mart, Dept. G • 2700 NE Andresen Road • Vancouver, WA 98661
Phone orders welcome: 206-695-1393
Same day shipping/No C.O.D. orders outside U.S.
CHECKS, MONEY ORDERS OR VISA/MASTERCARD.
PLEASE NOTE: Free shipping & handling on all orders • C.O.D. add $3.00 to total order • All orders must be paid in U.S. funds.

VISA

MasterCard

Complete Package

$39.95

*Details inside package